



# L<sup>A</sup>T<sub>E</sub>X AND FRIENDS

**M.R.C. VAN DONGEN**



© 2010 by M.R.C. van Dongen. All rights reserved.



# Preface

THIS BOOK is still in preparation. It provides computer science graduate (or equivalent) students with an introduction to technical writing and presenting with  $\text{\LaTeX}$ , which is the de-facto standard in computer science and mathematics. This includes techniques for writing large and complex documents and presentations as well as an introduction to the creation of complex graphics in an integrated manner.

I have tried to minimise the number of classes and style files which the students need to know. This is one of the main reasons why I decided to use the `amsmath` package for the presentation of mathematics, and `tikz`, `pgfplots`, and `beamer` for the creation of diagrams, graphs, and presentations. Another advantage of this approach is that this simplifies the process of creating a viewable/printable output file: everything should work with `pdflatex`.

Writing a document like this teaches you much about  $\text{\LaTeX}$ , which is why I intend to maintain two versions of this document. One version which can be used as an ultimate reference manual, and one slimmed down version which is intended for the students.

This being a preliminary version, with many chapters still pending or incomplete, any comments and suggestions about the presentation and new topics will be much appreciated.

M.R.C. van Dongen  
Cork  
2010





# Contents

<b>I</b>	<b>Basics</b>	<b>1</b>
<b>1</b>	<b>Introduction to L<sup>A</sup>T<sub>E</sub>X</b>	<b>3</b>
1.1	Pros and Cons . . . . .	4
1.2	Basics . . . . .	6
1.2.1	The T <sub>E</sub> X Processors . . . . .	6
1.2.2	From .tex to .dvi and Friends . . . . .	7
1.2.3	The Name of the Game . . . . .	8
1.2.4	Staying in Sync . . . . .	8
1.2.5	Writing a L <sup>A</sup> T <sub>E</sub> X Input Document . . . . .	8
1.2.6	The Abstract . . . . .	12
1.2.7	Spaces, Comments, and Paragraphs . . . . .	12
1.3	Document Hierarchy . . . . .	13
1.3.1	Minor Document Divisions . . . . .	14
1.3.2	Major Document Divisions . . . . .	15
1.3.3	The Appendix . . . . .	16
1.4	Document Management . . . . .	16
1.5	Labels and Cross-references . . . . .	17
1.6	Controlling the Style of References . . . . .	19
1.7	The Bibliography . . . . .	20
1.7.1	Basic Usage . . . . .	20
1.7.2	The bibtex Program . . . . .	24
1.7.3	The natbib Package . . . . .	26
1.7.4	Multiple Bibliographies . . . . .	28
1.7.5	Bibliographies at End of Chapter . . . . .	29
1.8	Reference Lists . . . . .	29
1.8.1	Table of Contents and Lists of Things . . . . .	29
1.8.2	Controlling the Table of Contents . . . . .	30
1.8.3	Controlling the Sectional Unit Numbering . . . . .	30
1.8.4	Indexes and Glossaries . . . . .	30
1.9	Class Files . . . . .	33
1.10	Packages . . . . .	34
1.11	Useful Classes and Packages . . . . .	35
1.12	Errors and Troubleshooting . . . . .	35

<b>II</b>	<b>Basic Typesetting</b>	<b>37</b>
<b>2</b>	<b>Running Text</b>	<b>39</b>
2.1	Special Characters . . . . .	39
2.1.1	Tieing Text . . . . .	40
2.1.2	Grouping . . . . .	41
2.2	Diacritics . . . . .	42
2.3	Ligatures . . . . .	43
2.4	Quotation Marks . . . . .	43
2.5	Dashes . . . . .	44
2.6	Periods . . . . .	45
2.7	Emphasis . . . . .	45
2.8	Footnotes and Marginal Notes . . . . .	46
2.9	Displayed Quotations and Verses . . . . .	47
2.10	Line Breaks . . . . .	47
2.11	Controlling the Size . . . . .	48
2.12	Controlling the Type Style . . . . .	49
2.13	Phantom Text . . . . .	49
2.14	Alignment . . . . .	50
2.14.1	Centred Text . . . . .	50
2.14.2	Flushed/Ragged Text . . . . .	51
2.14.3	Basic tabular Constructs . . . . .	51
2.14.4	The booktabs Package . . . . .	53
2.14.5	Advanced tabular Constructs . . . . .	54
2.14.6	The tabbing Environment . . . . .	56
2.15	Language Related Issues . . . . .	57
2.15.1	Hyphenation . . . . .	57
2.15.2	Foreign Languages . . . . .	58
2.15.3	Spell-Checking . . . . .	58
<b>3</b>	<b>Lists</b>	<b>59</b>
3.1	Unordered Lists . . . . .	59
3.2	Ordered Lists . . . . .	61
3.3	The enumerate Package . . . . .	61
3.4	Description Lists . . . . .	62
3.5	Making your Own Lists . . . . .	63
<b>III</b>	<b>Pictures, Diagrams, Tables, and Graphs</b>	<b>67</b>
<b>4</b>	<b>Presenting External Pictures</b>	<b>69</b>
4.1	The figure Environment . . . . .	69
4.2	Special Packages . . . . .	71
4.2.1	Floats . . . . .	71
4.2.2	Legends . . . . .	71
4.3	External Picture Files . . . . .	71
4.4	The graphicx Package . . . . .	72



4.5	Setting Default Key Values . . . . .	72
4.6	Setting a Search Path . . . . .	73
4.7	Defining Graphics Extensions . . . . .	73
4.8	Conversion Tools . . . . .	74
4.9	Defining Graphics Conversion . . . . .	74
<b>5</b>	<b>Presenting Diagrams with <code>tikz</code></b>	<b>75</b>
5.1	Why Specify your Diagrams? . . . . .	75
5.2	The <code>tikzpicture</code> Environment . . . . .	75
5.3	The <code>\tikz</code> Command . . . . .	76
5.4	Grids . . . . .	77
5.5	Paths . . . . .	77
5.6	Coordinate Labels . . . . .	78
5.7	Extending Paths . . . . .	79
5.8	Actions on Paths . . . . .	82
5.8.1	Colour . . . . .	83
5.8.2	Drawing the Path . . . . .	85
5.8.3	Line Width . . . . .	85
5.8.4	Line Cap and Join . . . . .	86
5.8.5	Dash Patterns . . . . .	87
5.8.6	Arrows . . . . .	88
5.8.7	Filling a Path . . . . .	89
5.8.8	Path Filling Rules . . . . .	90
5.9	Nodes and Node Labels . . . . .	91
5.9.1	Predefined Nodes Shapes . . . . .	92
5.9.2	Node Options . . . . .	93
5.9.3	Connecting Nodes . . . . .	95
5.9.4	Special Node Shapes . . . . .	95
5.10	Coordinate Systems . . . . .	97
5.11	Coordinate Calculations . . . . .	98
5.11.1	Relative and Incremental Coordinates . . . . .	99
5.11.2	Complex Coordinate Calculations . . . . .	100
5.12	Options . . . . .	102
5.13	Styles . . . . .	102
5.14	Scopes . . . . .	103
5.15	The <code>\foreach</code> Command . . . . .	104
5.16	The <code>let</code> Operation . . . . .	106
5.17	The <code>To Path</code> Operation . . . . .	107
5.18	The <code>spy</code> Library . . . . .	108
5.19	Trees . . . . .	108
5.20	Logical Circuits . . . . .	110
5.21	Installing <code>tikz</code> . . . . .	111
<b>6</b>	<b>Presenting Data with Tables</b>	<b>113</b>
6.1	The Purpose of Tables . . . . .	113
6.2	Kinds of Tables . . . . .	113
6.3	The Anatomy of Tables . . . . .	114

6.4	Designing Tables . . . . .	115
6.5	The <code>table</code> Environment . . . . .	118
6.6	Wide Tables . . . . .	119
6.7	Multi-page Tables . . . . .	119
6.8	Databases and Spreadsheets . . . . .	120
<b>7</b>	<b>Presenting Data with Graphs</b>	<b>123</b>
7.1	The Purpose of Graphs . . . . .	123
7.2	Pie Charts . . . . .	124
7.3	Introduction to <code>pgfplots</code> . . . . .	125
7.4	Bar Graphs . . . . .	126
7.5	Paired Bar Graphs . . . . .	128
7.6	Component Bar Graphs . . . . .	129
7.7	Coordinate Systems . . . . .	130
7.8	Line Graphs . . . . .	132
7.9	Scatter Plots . . . . .	134

## **IV Mathematics and Algorithms 137**

<b>8</b>	<b>Mathematics</b>	<b>139</b>
8.1	The $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{\LaTeX}$ Platform . . . . .	140
8.2	$\text{\LaTeX}$ 's Math Modes . . . . .	141
8.3	Ordinary Math Mode . . . . .	141
8.4	Subscripts and Superscripts . . . . .	142
8.5	Greek Letters . . . . .	142
8.6	Displayed Math Mode . . . . .	143
8.6.1	The <code>equation</code> Environment . . . . .	144
8.6.2	The <code>split</code> Environment . . . . .	145
8.6.3	The <code>multline</code> Environment . . . . .	146
8.6.4	The <code>gather</code> Environment . . . . .	147
8.6.5	The <code>align</code> Environment . . . . .	147
8.6.6	Intermezzo: Increasing Productivity . . . . .	148
8.6.7	Interrupting a Display . . . . .	149
8.6.8	Low-level Alignment Building Blocks . . . . .	149
8.6.9	The <code>eqnarray</code> Environment . . . . .	150
8.7	Text in Formulae . . . . .	150
8.8	Delimiters . . . . .	150
8.8.1	Scaling Left and Right Delimiters . . . . .	151
8.8.2	Bars . . . . .	152
8.8.3	Tuples . . . . .	152
8.8.4	Floors and Ceilings . . . . .	153
8.8.5	Delimiter Commands . . . . .	153
8.9	Fractions . . . . .	153
8.10	Sums, Products, and Friends . . . . .	154
8.10.1	Basic Typesetting Commands . . . . .	155
8.10.2	Overriding the Basic Typesetting Style . . . . .	156

8.10.3	Multi-line Limits . . . . .	157
8.11	Functions and Operators . . . . .	158
8.11.1	Existing Operators . . . . .	158
8.11.2	Declaring New Operators . . . . .	159
8.11.3	Managing Content with the <code>cool</code> Package . . .	160
8.12	Integration and Differentiation . . . . .	160
8.12.1	Integration . . . . .	160
8.12.2	Differentiation . . . . .	161
8.13	Roots . . . . .	161
8.14	Arrays and Matrices . . . . .	162
8.15	Math Mode Accents, Hats, and Other Decorations . .	163
8.16	Braces . . . . .	163
8.17	Case-based Definitions . . . . .	165
8.18	Function Definitions . . . . .	166
8.19	Theorems . . . . .	166
8.19.1	Ingredients of Theorems . . . . .	166
8.19.2	Theorem-like Styles . . . . .	167
8.19.3	Defining Theorem-like Environments . . . . .	168
8.19.4	Defining Theorem-like Styles . . . . .	169
8.19.5	Proofs . . . . .	170
8.20	Mathematical Punctuation . . . . .	170
8.21	Spacing and Linebreaks . . . . .	172
8.21.1	Line Breaks . . . . .	172
8.21.2	Conditions . . . . .	172
8.21.3	Physical Units . . . . .	173
8.21.4	Sets . . . . .	173
8.21.5	More Spacing Commands . . . . .	174
8.22	Changing the Style . . . . .	174
8.23	Symbol Tables . . . . .	175
8.23.1	Operation Symbols . . . . .	175
8.23.2	Relation Symbols . . . . .	175
8.23.3	Arrows . . . . .	175
8.23.4	Miscellaneous Symbols . . . . .	176
<b>9</b>	<b>Algorithms and Listings</b>	<b>179</b>
9.1	Typesetting Algorithms with <code>algorithm2e</code> . . . . .	179
9.1.1	Importing <code>algorithm2e</code> . . . . .	179
9.1.2	Basic Environments . . . . .	180
9.1.3	Describing Input and Output . . . . .	181
9.1.4	Conditional Statements . . . . .	182
9.1.5	The Switch Statement . . . . .	183
9.1.6	Iterative Statements . . . . .	184
9.1.7	Comments . . . . .	185
9.2	Typesetting Listings with the <code>listings</code> Package . . . .	186

<b>V</b>	<b>Automation</b>	<b>191</b>
<b>10</b>	<b>Commands and Environments</b>	<b>193</b>
10.1	Why use Commands . . . . .	193
10.2	User-defined Commands . . . . .	195
10.2.1	Defining Commands Without Arguments . .	195
10.2.2	Defining Commands With Arguments . . . .	196
10.2.3	Fragile and Robust Commands . . . . .	197
10.2.4	Defining Robust Commands . . . . .	198
10.3	The T <sub>E</sub> X Processors . . . . .	198
10.4	Commands and Arguments . . . . .	199
10.5	Defining Commands with T <sub>E</sub> X . . . . .	201
10.6	Tweaking Existing Commands with \let . . . . .	204
10.7	More than Nine Arguments . . . . .	204
10.8	Introduction to Environments . . . . .	205
10.9	Environment Definitions . . . . .	206
<b>11</b>	<b>Option Parsing</b>	<b>209</b>
11.1	Why Use a $\langle \text{Key} \rangle = \langle \text{Value} \rangle$ Interface? . . . . .	209
11.2	The keyval Package . . . . .	209
11.3	The keycommand Package . . . . .	211
<b>12</b>	<b>Branching</b>	<b>213</b>
12.1	Counters, Booleans, and Lengths . . . . .	213
12.1.1	Counters . . . . .	213
12.1.2	Booleans . . . . .	214
12.1.3	Lengths . . . . .	215
12.1.4	Scoping . . . . .	217
12.2	The ifthen Package . . . . .	217
12.3	The calc Package . . . . .	219
12.4	Looping . . . . .	219
12.5	Tail Recursion . . . . .	220
<b>13</b>	<b>User-defined Styles and Classes</b>	<b>221</b>
13.1	User-defined Style Files . . . . .	221
13.2	User-defined Class Files . . . . .	221
<b>VI</b>	<b>Miscellany</b>	<b>223</b>
<b>14</b>	<b>Beamer Presentations</b>	<b>225</b>
14.1	Frames . . . . .	225
14.2	Modal Presentations . . . . .	227
14.3	Incremental Presentations . . . . .	229
14.4	Visual Alerts . . . . .	231
14.5	Adding Some Style . . . . .	231

<b>15</b>	<b>Installing <math>\text{\LaTeX}</math> and Friends</b>	<b>237</b>
15.1	Installing $\text{\TeX}$ Live . . . . .	237
15.2	Configuring $\text{\TeX}$ Live . . . . .	238
15.2.1	Adjusting the <code>PATH</code> . . . . .	238
15.2.2	Configuring <code>TEXINPUTS</code> . . . . .	238
15.3	Installing Classes and Packages . . . . .	239
15.4	Installing $\text{\LaTeX}$ Fonts . . . . .	240
15.5	Installing Unix Fonts . . . . .	240
15.6	Using the <code>fontspec</code> Package . . . . .	240
15.7	Package Managers . . . . .	241
<b>16</b>	<b>Resources</b>	<b>243</b>
16.1	Books about $\text{\TeX}$ and $\text{\LaTeX}$ . . . . .	243
16.2	Bibliography Resources . . . . .	243
16.3	Articles by the $\text{\LaTeX}$ 3 Team . . . . .	243
16.4	$\text{\LaTeX}$ Articles, Course Notes and Tutorials . . . . .	244
16.5	<code>METAPOST</code> Articles and Tutorials . . . . .	244
16.6	On-line Resources . . . . .	244
16.7	YouTube Resources . . . . .	245
16.8	English . . . . .	246
<b>VII</b>	<b>References and Bibliography</b>	<b>247</b>
	<b>Indices</b>	<b>249</b>
	Index of $\text{\LaTeX}$ and $\text{\TeX}$ Commands . . . . .	250
	Index of Environments . . . . .	259
	Index of Classes . . . . .	261
	Index of Packages . . . . .	262
	Index of Commands and Languages . . . . .	263
	<b>Acronyms</b>	<b>265</b>
	<b>Bibliography</b>	<b>267</b>



# List of Figures

1.1	Typical L <sup>A</sup> T <sub>E</sub> X program . . . . .	10
1.2	Defining comments . . . . .	13
1.3	Using \includeonly and \include. . . . .	17
1.4	Using \label and \ref. . . . .	18
1.5	Using \pageref. . . . .	18
1.6	Using the prettyref package. . . . .	20
1.7	A minimal bibliography. . . . .	21
1.8	The \cite command. . . . .	22
1.9	Using \cite with an optional argument. . . . .	23
1.10	Including a bibliography. . . . .	25
1.11	Some BIB <sub>T</sub> E <sub>X</sub> entries. . . . .	25
1.12	The \citet and \citep commands. . . . .	27
1.13	Using the \citeauthor and \citeyear commands. . . . .	27
1.14	Minimal letter. . . . .	34
2.1	Quotes. . . . .	44
2.2	Nested quotations. . . . .	44
2.3	Dashes . . . . .	45
2.4	Using footnotes. . . . .	46
2.5	The quote environment. . . . .	47
2.6	The verse environment. . . . .	48
2.7	Controlling the size. . . . .	49
2.8	The \phantom command. . . . .	50
2.9	Using the tabular environment. . . . .	53
2.10	Using booktabs rules. . . . .	54
2.11	Controlling column widths with an @-expression. . . . .	55
2.12	The tabbing environment. . . . .	57
2.13	Advanced use of tabbing environment. . . . .	57
2.14	Using the babel package. . . . .	58
3.1	The itemize environment. . . . .	60
3.2	Changing the item label. . . . .	60
3.3	The enumerate environment. . . . .	61
3.4	Using the enumerate package. . . . .	62
3.5	Using the description environment. . . . .	62
3.6	list format affecting lengths. . . . .	63
3.7	A user-defined list. . . . .	64
3.8	A user-defined environment for lists. . . . .	64

4.1	Using the <code>dpfloat</code> package. . . . .	71
4.2	Including an external graphics file. . . . .	72
5.1	Drawing a grid. . . . .	77
5.2	Creating a path. . . . .	78
5.3	Cubic spline in <code>tikz</code> . . . . .	80
5.4	Using a dash pattern . . . . .	87
5.5	Using a dash phase. . . . .	87
5.6	The ‘even odd rule’. . . . .	91
5.7	The ‘nonzero rule’. . . . .	91
5.8	Nodes and implicit labels. . . . .	92
5.9	Low-level node control. . . . .	93
5.10	Node placement. . . . .	95
5.11	Drawing lines between node shapes. . . . .	95
5.12	The <code>circle split</code> node style . . . . .	96
5.13	A node with <code>rectangle</code> style and several parts. . . . .	96
5.14	Using four coordinate systems. . . . .	98
5.15	Computing the intersection of perpendicular lines. . . . .	98
5.16	Absolute, relative, and incremental coordinates. . . . .	99
5.17	Coordinate computations with partway modifiers. . . . .	100
5.18	Coordinate computations with partway and distance modifiers. . . . .	100
5.19	Coordinate computations with projection modifiers. . . . .	100
5.20	Predefining options with the <code>\tikzset</code> command. . . . .	103
5.21	Using scopes . . . . .	104
5.22	The <code>\foreach</code> command. . . . .	105
5.23	Simple <code>to path</code> example. . . . .	107
5.24	A User-defined ‘ <code>to path</code> ’ style. . . . .	108
5.25	Using the <code>spy</code> library. . . . .	108
5.26	Drawing a tree. . . . .	109
5.27	Using implicit node labels in tree. . . . .	109
5.28	Controlling the node style. . . . .	110
5.29	Missing tree nodes. . . . .	110
5.30	Drawing a half adder with <code>tikz</code> . . . . .	112
6.1	Components of a demonstration table. . . . .	114
6.2	Creating a table with the <code>booktabs</code> package. . . . .	118
6.3	Using the <code>longtable</code> package. . . . .	120
7.1	A pie chart. . . . .	124
7.2	Using the <code>axis</code> environment. . . . .	125
7.3	Sample output of the <code>axis</code> environment. . . . .	125
7.4	A bar graph. . . . .	127
7.5	Creating a bar graph. . . . .	127
7.6	A paired bar graph. . . . .	129
7.7	Creating a paired bar graph. . . . .	129
7.8	A component bar graph. . . . .	131



7.9	Creating a component bar graph. . . . .	131
7.10	A line graph. . . . .	132
7.11	Creating a line graph. . . . .	133
7.12	A scatter plot. . . . .	134
7.13	Creating a scatter plot. . . . .	134
8.1	The <code>\shortintertext</code> command. . . . .	149
8.2	The <code>aligned</code> environment. . . . .	149
8.3	'Limit' argument of log-like functions. . . . .	158
8.4	Using the <code>amsthm</code> package. . . . .	169
8.5	Using the mathematical punctuation commands. . . . .	171
9.1	Effect of the options <code>noline</code> , <code>lined</code> , and <code>vlined</code> . . . . .	180
9.2	Using <code>algorithm2e</code> . . . . .	181
9.3	Typesetting conditional statements with <code>algorithm2e</code> . . . . .	183
9.4	Using <code>algorithm2e</code> 's <code>switch</code> statements. . . . .	184
9.5	Creating a partial listing with the <code>listings</code> package. . . . .	187
9.6	Listing resulting from Figure 9.5 . . . . .	187
9.7	Setting new defaults with the <code>\lstset</code> command. . . . .	188
10.1	User-defined commands. . . . .	197
10.2	A program with user-defined combinators. . . . .	200
10.3	Defining commands with default arguments. . . . .	203
10.4	A <code>sectionl</code> unit environment. . . . .	205
10.5	Using more than nine arguments. . . . .	205
10.6	User-defined environment. . . . .	206
12.1	A tail recursion-based implementation of a lisp-like <code>\apply</code> command. . . . .	220
14.1	Creating a titlepage with the <code>beamer</code> class. . . . .	226
14.2	The <code>frame</code> title commands. . . . .	227
14.3	Using the <code>beamerarticle</code> package. . . . .	228
14.4	Using modes. . . . .	228
14.5	Using the <code>\pause</code> command. . . . .	230
14.6	Using overlay specifications. . . . .	230
14.7	Adding visual alerts. . . . .	231
14.8	Using a <code>beamer</code> theme. . . . .	232
14.9	Sample output of <code>beamer</code> 's default theme. . . . .	233
14.10	Sample output of <code>beamer</code> 's <code>Boadilla</code> theme. . . . .	233
14.11	Sample output of <code>beamer</code> 's <code>Antibes</code> theme. . . . .	234
14.12	Sample output of <code>beamer</code> 's <code>Goettingen</code> theme. . . . .	234
15.1	Using the <code>fontspec</code> package. . . . .	241



# List of Tables

1.1	Depth values of sectional unit commands. . . . .	30
1.2	Using the <code>\index</code> command. . . . .	32
2.1	Ten special characters. . . . .	40
2.2	Common diacritics. . . . .	42
2.3	Other special characters. . . . .	43
2.4	Foreign ligatures. . . . .	43
2.5	Size-affecting declarations and environments. . .	48
2.6	Type style affecting declarations and commands. .	49
2.7	Using <code>booktabs</code> rules. . . . .	54
5.1	The <code>xcolor</code> colours. . . . .	83
5.2	Arrow head types. . . . .	89
5.3	Shorthand notation for the <code>\foreach</code> command. .	105
5.4	Node shapes provided by logic gate shape libraries. .	111
6.1	A poorly designed table. . . . .	115
6.2	An improved version of Table 6.1. . . . .	117
7.1	Allowed values for <code>mark</code> option. . . . .	135
8.1	Lowercase Greek letters. . . . .	143
8.2	Uppercase Greek letters. . . . .	143
8.3	Variable-size delimiters. . . . .	154
8.4	Variable-sized symbols. . . . .	156
8.5	Log-like functions. . . . .	158
8.6	Integration signs. . . . .	161
8.7	Math mode accents, hats, and other decorations. .	164
8.8	Math mode dot-like symbols. . . . .	170
8.9	Positive and negative spacing. . . . .	174
8.10	Binary operation symbols. . . . .	175
8.11	Relation symbols. . . . .	176
8.12	Additional <code>amsmath</code> -provided relation symbols. .	176
8.13	Fixed-size arrow symbols. . . . .	177
8.14	Extensible <code>amsmath</code> -provided arrow symbols. . .	177
8.15	Extensible <code>mathtools</code> -provided arrow symbols. .	177
8.16	extensible <code>mathtools</code> -provided arrow symbols. .	178
8.17	Miscellaneous symbols. . . . .	178
10.1	How expansion works. . . . .	201

12.1 Length units. . . . .	215
----------------------------	-----

# **Part I**

## **Basics**



# CHAPTER I

## Introduction to L<sup>A</sup>T<sub>E</sub>X

THIS CHAPTER is an introduction to L<sup>A</sup>T<sub>E</sub>X and friends but it is *not* about typesetting fancy things. Typesetting fancy things is dealt with in subsequent chapters. The main purpose of this chapter is to provide an understanding of the *basic* mechanisms of L<sup>A</sup>T<sub>E</sub>X, using plain text as a vehicle. After reading this chapter you should know how to:

- Write a simple L<sup>A</sup>T<sub>E</sub>X input document based on the `article` class.
- Turn the input document into `.pdf` with the aid of the `pdflatex` program.
- Define *labels* and use them to create consistent cross-references to chapter and sections. This basic cross-referencing mechanism also works for tables, figures, and so on.
- Create a fault-free table of contents with the `\tableofcontents` command. Creating a list of tables and a list of figures works in a similar way.
- Cite the literature with the aid of the `\cite` command.
- Generate one or several bibliographies from your citations with the `bibtex` program.
- Change the appearance of the bibliographies by choosing the proper bibliography style.
- Manage the structure and writing of your document by exploiting the `\include` command.
- Control the visual presentation of your article by selecting the right `article` class options.
- Much, much, more.

**Intermezzo.** *L<sup>A</sup>T<sub>E</sub>X gives you output documents which looks great and have consistent cross-references and citations. Much of the creation of the output documents is automated and done behind the scenes. This gives you extra time to think about the ideas you want to present and how to communicate these ideas in an effective way. One way to communicate effectively is planning: the order and the purpose of the writing determines how it is received by your target audience. L<sup>A</sup>T<sub>E</sub>X's markup helps you concretise the purpose of your writing and present it in a consistent manner. As a matter of fact, L<sup>A</sup>T<sub>E</sub>X forces you to think about the purpose of your writing and this improves the effectiveness of the presentation of your ideas. All that's left to*

*you is determine the order of presentation and provide some extra markup. To determine the order of your presentation and to write your document you can treat  $\text{\LaTeX}$  as a programming language. This means that you can use software engineering techniques such as top-down design and stepwise refinement. These techniques may also help when you haven't completely figured out what it is you want to write.*

Throughout this chapter it is assumed that you are using the `Unix` (Linux: `ubuntu`, `debian`, ...) operating system. Time permitting, a section will be added on how to run  $\text{\LaTeX}$  on different operating systems.

## 1.1 Pros and Cons

Before we start, it is good to look at arguments in favour of  $\text{\LaTeX}$  and arguments against it. Some of these arguments are based on <http://nitens.org/taraborelli/latex>.

**Cons** The following are some common and less common arguments against  $\text{\LaTeX}$ .

- $\text{\LaTeX}$  is difficult. It may take one to several months to learn. True, learning  $\text{\LaTeX}$  does take a while. However, it will save you time in the long run, even if you're writing a minor thesis.
- $\text{\LaTeX}$  is not a What You See is What You Get (WYSIWYG) word-processor. Correct, but there are many  $\text{\LaTeX}$  Integrated Development Environments (IDES) and some IDEs such as `eclipse` have  $\text{\LaTeX}$  plugins.
- There is little support for physical markup. Yes, but for most papers, notes, and theses in computer science, mathematics, and other technical and non-technical fields, there are existing packages which you can use without having to fiddle with the way things look. However, if you really need to tweak the output then you may have to put in extra time, which may slow down the writing. Then again, you should be able to reuse this effort for other projects.
- Using non-standard fonts is difficult. This used to be true. However, with the arrival of the `fontspec` package and `xelatex` using non-standard fonts is easy. Furthermore, it is more than likely that for most day-to-day work you wouldn't *want* any non-standard fonts.
- It takes some practice to let text flow around pictures. That's a tricky one. Usually, you let  $\text{\LaTeX}$  determine the positions of your figures. As a consequence they may not always end up where you intended them to be. Sometimes the text in the vicinity of such figures doesn't look nice: the text doesn't flow. You can improve the text flow by rearranging a few words in adjacent paragraphs but this *does* take some practice.



- There are too many L<sup>A</sup>T<sub>E</sub>X packages, which makes it difficult to find the right package. Agreed, but most L<sup>A</sup>T<sub>E</sub>X documents require only a few core packages, which are easy to find. Moreover, asking a question in the mailing list `comp.text.tex` usually results in some quick pointers. You may also find this list at <http://groups.google.com/group/comp.text.tex/topics>.
- L<sup>A</sup>T<sub>E</sub>X encourages structured writing and the separation of style from content, which is not how many people (especially non-programmers) work. Well, it seems times they are-a-changin' because more and more new (new?) communities have started using L<sup>A</sup>T<sub>E</sub>X [Burt, 2005; Thomson, 2008a; Thomson, 2008b; Buchsbaum and Reinaldo, 2007; Garcia and Buchsbaum, 2010; Breitenbucher, 2005; Senthil, 2007; Dearborn, 2006; Veytsman and Akhmadeeva, 2006]. Some communities have organised and have created their own websites: <http://theotex.blogspot.com/>, <http://www-lmbb.ncifcrf.gov/~toms/latex.html>, <https://coral.uchicago.edu:8443/display/humcomp/LaTeX>, and <http://www.essex.ac.uk/linguistics/external/clmt/latex4ling/>.

**Pros** The following are arguments in favour of L<sup>A</sup>T<sub>E</sub>X:

- L<sup>A</sup>T<sub>E</sub>X provides state-of-the art typesetting, including kerning, real small caps, common and non-common ligatures, glyph variants, .... It also does a very good job at automated hyphenation.
- Many conferences and publishers accept L<sup>A</sup>T<sub>E</sub>X. In addition they provide style and class files which guarantee documents conforming to the required formatting guidelines.
- L<sup>A</sup>T<sub>E</sub>X is a Turing-complete programming language. This gives you almost complete control. For example, you can decide which things should be typeset and how this should be done.
- With L<sup>A</sup>T<sub>E</sub>X you can prepare several documents from the same source file. Not only lets this control you which text should be used in which document but also how it should appear.
- L<sup>A</sup>T<sub>E</sub>X is highly configurable. Changing the appearance of your document is done by choosing the proper document class, class options, packages, and package options. The proper use of commands supports consistent appearance and gives you ultimate control.
- You can translate L<sup>A</sup>T<sub>E</sub>X to html/ps/pdf/DocBook ....
- L<sup>A</sup>T<sub>E</sub>X automatically numbers your chapters, sections, figures, and so on. In addition it provides cross-referencing support.
- L<sup>A</sup>T<sub>E</sub>X has excellent bibliography support. It supports consistent citations and an automatically generated bibliography with a consistent look and feel. The style of citations and the organisation of the bibliography is configurable.
- There is some support for WYSIWYG document preparation: lyx (<http://www.lyx.org/>), T<sub>E</sub>Xmacs (<http://www.texmacs.org/>), .... Furthermore, some editors and IDEs provide support for L<sup>A</sup>T<sub>E</sub>X,

e.g., vim, emacs, eclipse, ....

- $\text{\LaTeX}$  is *very* stable, free, and available on many platforms.
- There is a very large, active, and helpful  $\text{\TeX}/\text{\LaTeX}$  user-base. Good starting points are listed in Section 16.6.
- $\text{\LaTeX}$  has comments.
- There's a very useful package which produces coffee stains on your papers [ *$\text{\LaTeX}$  Coffee Stains*]. Again, this adds consistency to your output documents.
- Most importantly:  $\text{\LaTeX}$  is fun!

## 1.2 Basics

$\text{\LaTeX}$  [Lamport, 1994] was written by Leslie Lamport as an extension of Donald Knuth's  $\text{\TeX}$  program [Knuth, 1990]. It consists of a Turing-complete procedural markup language and a typesetting processor. The combination of the two lets you control both the visual presentation *as well as* the content of your documents. The following three steps explain how you use  $\text{\LaTeX}$ .

1. You write your document in a  $\text{\LaTeX}$  (`.tex`) input (source) file.
2. You run the `latex` program on your input file. This turns the input file into a *device independent file* (a `.dvi` file). Depending on your source file there may be errors which you may have to fix before you can continue to the final step.
3. You view the `.dvi` file on your computer or convert it to another format (usually a printable document format).

### 1.2.1 The $\text{\TeX}$ Processors

Roughly speaking  $\text{\LaTeX}$  is built on top of  $\text{\TeX}$ . This adds extra functionality to  $\text{\TeX}$  and makes writing your document much easier.  $\text{\LaTeX}$  being built on top of  $\text{\TeX}$ , the result is a  $\text{\TeX}$  program. You may get a good understanding of  $\text{\LaTeX}$  by studying  $\text{\TeX}$ 's four *processors*, which are basically run in a “pipeline” [Knuth, 1990; Eijkhout, 2007; Abrahams, Hargreaves and Berry, 2003]. The following are the main functions of  $\text{\TeX}$ 's processors.

**Input Processor** Turns source file into a token stream. The resulting token stream is sent to the Expansion Processor.

**Expansion Processor** Turns token stream into token stream. Expandable tokens are repeatedly expanded until there are no more left. The expansion applies to commands, conditionals, and some primitive commands. The resulting output is sent to the Execution Processor.

**Execution Processor** Executes executable control sequences. These actions may affect the state. This applies, for example, to assignments and command definitions. The Execution Processor also

constructs horizontal, vertical, and mathematical lists. The final output is sent to the Visual Processor.

**Visual Processor** Creates `.dvi` file. This is the final stage. It turns horizontal lists into paragraphs, breaks vertical lists into pages, and turns mathematical lists into formulae.

### 1.2.2 From `.tex` to `.dvi` and Friends

Now that you know a bit about how L<sup>A</sup>T<sub>E</sub>X works, it's time to study the programs you need to turn your input files into readable output. You may ignore this section if you use an IDE because your IDE will do all the necessary things to create your output file, without the need for user intervention at the command-line level.

In its simplest form the `latex` program turns your L<sup>A</sup>T<sub>E</sub>X input file into the device independent file (`.dvi` file), which you can view and turn into other output formats, including `.pdf`. Before going into the details about the L<sup>A</sup>T<sub>E</sub>X syntax, let's see how you turn an existing L<sup>A</sup>T<sub>E</sub>X source file into a `.dvi` file. To this end, let's assume you have an error-free L<sup>A</sup>T<sub>E</sub>X source file which is called '`<base name>.tex`'. The following command turns your source file into an output file called '`<base name>.dvi`'.

```
$ latex <base name>.tex
```

Unix Session

With `latex` you may omit the `.tex` extension.

```
$ latex <base name>
```

Unix Session

The resulting `.dvi` output can be viewed with the `xdvi` program.

```
$ xdvi <base name>.dvi &
```

Unix Session

Now that you have the `.dvi` version of your L<sup>A</sup>T<sub>E</sub>X program, you may convert it to other formats. The following converts `<base name>.dvi` to postscript (`<base name>.ps`).

```
$ dvips -o <base name>.ps <base name>.dvi
```

Unix Session

The following converts `<base name>.dvi` to portable document format (`.pdf`).

```
$ dvi2pdf <base name>.dvi
```

Unix Session

However, by far the easiest to generate `.pdf` is using the `pdflatex` program. As with `latex`, `pdflatex` does not need the `.tex` extension.

```
$ pdflatex <base name>.tex
```

Unix Session

**Intermezzo.** *If you're writing a book, a thesis, or an article then generating `.dvi` and viewing it with `xdvi` is by far the quickest. However, there may be problems with graphics, which may not always be rendered properly. I find it convenient to (1) run the `xdvi` program in the background (using the `&` operator), (2) position the `xdvi` window over the terminal window which I'm using to edit the L<sup>A</sup>T<sub>E</sub>X program, and (3) edit the program with `vim`. You*

*can execute shell commands from within vim by going to command mode and issuing the command: '`<ESC>: !<command><RETURN>`' to execute the command `<command>` or '`<ESC>: !!<RETURN>`' for the most recently executed command.<sup>1</sup> This lets you run latex from within your editor on the file you're editing. Most Linux Graphical User Interfaces (GUIs) let you cycle from window to window by typing a magic spell: in KDE it is '`<ALT><TAB>`'. Typing this spell lets me quickly cycle from my editing session to the viewing sessions and back. Using this mechanism keeps my hands on the keyboard and saves time, wrists, and elbows.*

### 1.2.3 The Name of the Game

Just like C, lisp, pascal, java, and other programming languages,  $\text{\LaTeX}$  may be viewed as a program or a language. When referring to the language this book usually uses  $\text{\LaTeX}$  and when referring to the program it usually writes latex. However, when writing latex the book actually means pdflatex because this is by far the easiest way to create viewable and printable output. Finally, when this book uses  $\text{\LaTeX}$  program it usually means  $\text{\LaTeX}$  source file.

### 1.2.4 Staying in Sync

The latex program sometimes needs more than a single run before it produces its final output. The following explains what happens when you and latex are no longer in sync.

To create a perfect output file and have consistent cross-references and citations, latex also writes information to and reads information from *auxiliary* files. Auxiliary files contain information about page numbers of chapters, sections, tables, figure, and so on. Some auxiliary files are generated by latex itself (e.g., .aux files). Others are generated by external programs such as bibtex, which is a program that generates information for the bibliography. When an auxiliary file changes then  $\text{\LaTeX}$  may be out of sync. You should rerun latex when this happens. Normally, latex outputs a warning when it suspects this is required:

```
$ latex document.tex
... LaTeX Warning: Label(s) may have changed. ...
Rerun to get cross-references right.
$
```

### 1.2.5 Writing a $\text{\LaTeX}$ Input Document

$\text{\LaTeX}$  is a markup language and document preparation system. It forces you to focus on the content and *not* on the presentation. In a  $\text{\LaTeX}$  program you write the content of your document, you use commands to provide markup and automate tasks, and you import libraries. The following explains this in further detail.

<sup>1</sup>The emacs program should let you to do similar things.

**Content** The content of your document is determined in terms of text and logical markup. L<sup>A</sup>T<sub>E</sub>X forces you to focus on the logical structure of your document. You provide this structure as markup in terms of familiar notions such as the author of the document, the title of a section, the body and the caption of a figure, the start and end of a list, the items in the list, a mathematical formula, a theorem, a proof, ....

**Commands** The main purpose of commands is to provide markup. For example, to specify the author of the document you write ‘\author{`<author name>`}’. The real strength of L<sup>A</sup>T<sub>E</sub>X is that it also is a Turing-complete programming language which lets you define your own commands. These commands let you do real programming and give you ultimate control over the content and the final visual presentation. You can reuse your commands by putting them in a library.

**Libraries** There are many existing document classes and packages (style files). Class files define rules which determine the appearance of the output document. They also provide the required markup commands. Packages are best viewed as libraries. They provide useful commands which automate many tedious tasks. However, some packages may affect the appearance of the output document.

Throughout this book, L<sup>A</sup>T<sub>E</sub>X input is typeset in a style which is reminiscent of the layout of a computer programming language input file. The style is very generous when it comes to inserting redundant space characters. Whilst not strictly necessary, this input layout has several advantages:

**Recognise structure** Carefully formatting your input helps you recognise the structure of your L<sup>A</sup>T<sub>E</sub>X source files. This makes it easier to locate the start and end of sentences and higher-level building blocks such as *environments*. (Environments are explained further on.)

**Mimic output** By formatting the input you can mimic the output. For example when you design a table with rows and columns you can align the columns in the input. This makes it easier to design the output.

**Find errors** This is related to the previous item. Formatting may help you find the cause of errors more quickly. For example, you can reduce the number of candidate error locations by commenting out entire lines. This is much easier than commenting out parts of lines, which usually requires many more editing operations. Especially if your editor supports “multiple undo/redone operations” this makes locating the cause of errors very easy.

**Figure 1.1**  
Typical  $\text{\LaTeX}$  program

---

```

\documentclass[a4paper,11pt]{article}

%Use the mathptmx package.
\usepackage{mathptmx}

\author{A.~U.~Thor}
\title{Introduction to \LaTeX}
\date{\today}

\begin{document} %Here we go.
  \maketitle
  \section{Introduction}
  \textvisiblespace The start.
  \section{Conclusion}
  \textvisiblespace The end.
\end{document}

```

---

Figure 1.1 depicts a typical example of a  $\text{\LaTeX}$  input program. For this example all spaces in the input have been made explicit by typesetting them with the symbol ‘ $\text{\textvisiblespace}$ ’, which represents a single space. The symbol ‘ $\text{\textvisiblespace}$ ’ is called *visible space*. In case you’re wondering, the command `\textvisiblespace` typesets the visible space.

The remainder of this section studies the example program in more detail. Spaces are no longer made explicit.

The third line in the input program is a comment. Comments start with a percentage sign (%) and last until the end of the line. Comments, as is demonstrated in the input program, may also start in the middle of a line.

The following command tells  $\text{\LaTeX}$  that your document should be typeset using the rules determined by the `article` document class.

```
\documentclass[a4paper,11pt]{article}
```

*$\text{\LaTeX}$  Input*

You can only have one document class per  $\text{\LaTeX}$  source file. The `\documentclass` command determines the document class. The command takes one *required* argument, which may be a single character or a sequence of characters inside braces (curly brackets). The argument is the name of the document class. In our example the required argument is `article` so the document class is `article`. You usually use the `\documentclass` command on the first line of your  $\text{\LaTeX}$  input file.

In our example, the `\documentclass` command also takes an *optional* argument. An optional argument is passed inside the square brackets immediately after the command (this is standard). Optional arguments are called *optional* because they may be omitted. If you omit them then you should omit the square brackets. In our example the ‘`a4paper,11pt`’ are options of the `\documentclass` command. The `\documentclass` command passes these options to the `article` class. This sets the default

page size to A4 with wide margins and sets the font size to 11 point.

The following command includes a package called `mathptmx`.

```
\usepackage{mathptmx}
```

L<sup>A</sup>T<sub>E</sub>X Input

The `mathptmx` package sets the default font to *Times Roman*. This is a very compact font, which may save you precious pages in the final document. Using the font is especially useful when you're fighting against page limits.

Packages may also take options. This works just as with document classes. You pass the options to the package by including them in square brackets after the `\usepackage` command.

The following three commands, which are best used in the preamble of the input document, are logical markup commands. These commands do not produce any output but they define the author, title, and date of our article.

```
\author{A.~U. Thor}
\title{Introduction to \LaTeX}
\date{\today}
```

L<sup>A</sup>T<sub>E</sub>X Input

The command `\LaTeX` in the argument of the `\title` command is for typesetting L<sup>A</sup>T<sub>E</sub>X. The purpose of the *tilde* (`~`) is explained further on in this chapter. For the moment you may assume that it typesets a single space.

The title is typeset by the `\maketitle` command. Usually, you put this command at the start of the document *environment*, which is the text between the `\begin{document}` and the `\end{document}`. You separate author names with the `\and` command in the argument of the `\author` command:

```
\author{T.~Dee \and T.~Dum}
```

L<sup>A</sup>T<sub>E</sub>X Input

You acknowledge friends, colleagues, and funding institutions by including a `\thanks` command as part of the argument of the `\author` command. This produces a footnote consisting of the argument of the `\thanks` command.

```
\author{Sinead\thanks{You're a lovely audience.}}
```

L<sup>A</sup>T<sub>E</sub>X Input

If you wish to build your own titlepage, then you may do this with the `titlepage` environment. This environment gives you complete control *and* responsibility. The `\titlepage` command and the `titlepage` environment may only be used after the `\begin{document}`.

```
\begin{document}
  \begin{titlepage}
    ...
  \end{titlepage}
  :
\end{document}
```

L<sup>A</sup>T<sub>E</sub>X Input

For the `article` class, as well as for most other L<sup>A</sup>T<sub>E</sub>X classes, you write the main text of the document in the document *environment*. This

environment starts with ‘`\begin{document}`’ and ends in ‘`\end{document}`’. We say that text is “in” the document environment if it is between the ‘`\begin{document}`’ and ‘`\end{document}`’. The text before ‘`\begin{document}`’ is called the *preamble* of the document. Sometimes we call the text which is in the document environment the *body* of the document.

Definitions and configurations should be provided in the preamble. The text in the document environment defines the content. In the body of your document you may use the commands that are defined in the preamble. (More generally, you may define commands almost anywhere. You may use them as soon as they’re defined.)

The body of the document environment in the following example defines a rather empty document consisting of a title, two sections, and two sentences. The title is generated by the `\maketitle` command. The sections are defined with the `\section` command. Each section contains one sentence. The text ‘The start.’ is in the first section. The text ‘The end.’ is in the last.

```
\begin{document} % Here we go.
  \maketitle
  \section{Introduction}
    The start.
  \section{Conclusion}
    The end.
\end{document}
```

### 1.2.6 The Abstract

Many documents have an *abstract*, which is a short piece of text describing what is in the document. Typically, the abstract consists of a few lines and a few hundred words. You specify the abstract as follows.

```
\begin{abstract}
  This document is an introduction to \LaTeX.
  ...
\end{abstract}
```

In an article the abstract is typically positioned immediately after the `\maketitle` command. Abstracts in books are usually found on a page of their own.

Some class files may provide an `\abstract` command that defines the abstract. These class files may require that you use the `\abstract` command in the document preamble. The position of the abstract in the output file is determined by the class.

### 1.2.7 Spaces, Comments, and Paragraphs

The paragraph is one of the most important basic building blocks of your document. The paragraph formation rules depend on how `latex` treats spaces, empty lines, and comments. Roughly, the rules are as follows.<sup>2</sup>

<sup>2</sup>Here it is assumed that the text does not contain any commands.



**Figure 1.2**  
Defining comments

This is the first sentence  
of the first paragraph.

The second sentence of this  
paragraph ends in the word  
'elephant'.

This is the first sentence  
of the second paragraph.  
pa%comment

ragraph.  
The second sentence of this  
paragraph  
ends in the word 'eleph  
ant'.

This is the first sentence of  
the first paragraph. The sec-  
ond sentence of this paragraph  
ends in the word 'elephant'.

This is the first sentence of the  
second paragraph. The sec-  
ond sentence of this paragraph  
ends in the word 'ant'.

In its default mode, latex treats a sequence of more than one space as if it were a single space. The end of line is the same as a space. However:

- An empty line acts as an end-of-paragraph specifier.
- A percentage character (%) starts a comment which ends at the end of the line.
- Spaces at the start of a line following a comment are ignored.

If you understand the example in Figure 1.2 then you probably understand these rules. In this example, the input is to the left and the resulting output to the right. This convention is used throughout this book, except for Chapter 5, which presents pictures to the left and L<sup>A</sup>T<sub>E</sub>X input to the right.

## 1.3 Document Hierarchy

The coarse-level logical structure of your document is formed by the parts in the document, chapters in parts, sections in chapters, subsections in sections, subsubsection in subsections, paragraphs, and so on. This defines the *document hierarchy*. Following [Lamport, 1994], we shall refer to the members of the hierarchy as *sectional units*.

**Intermezzo.** *The sectional units are crucial for presenting effectively. For example, you break down the presentation of a thesis by giving it chapters. The chapters should be ordered to ease the flow of reading. The titles of the chapters are also important. Ideally chapter titles should be short, but most importantly each chapter title should describe what's in its chapter. To the reader a chapter title is a great help because it prepares them for what's in the chapter which they're about to read. A good chapter title is like an ultimate summary of the chapter. It prepares the reader's mindset and helps them digest what's in the chapter. If you are a student writing a thesis then good chapter titles are also important because they demonstrate your writing intentions.*

*Within chapters you present your sections in a similar way, by carefully breaking down what's in the chapter, by carefully arranging the order, and by carefully providing proper section titles. And so on.*

### 1.3.1 Minor Document Divisions

LaTeX provides the following sectional units:

**part** Optional unit which is used for major divisions.

**chapter** A chapter in a book or report.

**sections** A section, subsection, or subsubsection.

**paragraph** A named paragraph. Here paragraph is a small unit in a section.

**subparagraph** A named subparagraph. Here subparagraph is a small unit in a paragraph.

None of these sectional units are available in the `letter` class. LaTeX provides a command for each sectional unit that marks the start and the title of the sectional unit. The following shows how to define a chapter called 'Foundations' and a section called 'Notation'. The remaining commands work analogously.

```
\chapter{Foundations}
\section{Notation}
```

LaTeX Input

When LaTeX processes your document it numbers the sectional units. In its default mode it will output these numbers before the titles. For example, this section, which has the title 'Document Hierarchy', has the number 1.3. LaTeX also supplies *starred* versions of the sectional commands. These commands suppress the numbers of the sectional units. They are called starred versions because their names end in an asterisk (\*). The following is an example of the starred versions of the `\chapter` and `\section` commands.

```
\chapter*{Main Theorems}
\section*{A Useful Lemma}
```

LaTeX Input

All sectional unit commands take an optional argument. If you supply this argument it replaces the title of the sectional unit in the table of contents. This is useful if the real title is very long.

```
\chapter[Going to Wales]%
{My Amusing Adventures in
 L{}lanfairpwll{}lgwynyl{}lgogerychw%
 yrndrobwl{}l{}l{}lantysiliogogoch}
```

LaTeX Input

### 1.3.2 Major Document Divisions

Books and theses typically consist of *front matter*, *main matter*, and *back matter*. Some journal or conference article styles also require front, main, and back matter. The following is based on [Lamport, 1994, Page 80].

**Front matter** Main information about the document: a half and main title page, copyright page, preface or foreword, table of contents, ....

**Main matter** The main body of the document.

**Back matter** Further information about the document and other sources of information: index, afterword, bibliography, acknowledgements, colophon, ....

The commands `\frontmatter`, `\mainmatter`, and `\backmatter` indicate the start of the front, main, and back matter. The following artificial example shows how they may be used. Notice that the example does not include any text.

```
\begin{document}
  \frontmatter
    \maketitle
    \tableofcontents
  \mainmatter
    \chapter{Introduction}
    \chapter{Conclusion}
  \backmatter
    \chapter*{Acknowledgement}
    \addcontentsline{toc}{chapter}{\bibname}
    \bibliography{db}
\end{document}
```

In the example, the command `\bibliography` inserts the bibliography. The command is explained in Section 1.7. The command `\addcontentsline` inserts an entry for the bibliography in the table of contents.

Notice that the layout of the L<sup>A</sup>T<sub>E</sub>X program is such that it gives you a good overview of the structure of the program.

**Intermezzo.** *If you are writing a thesis then you should consider starting by writing down the chapter titles of your thesis first. Your titles should be good and, most importantly, they should be self-descriptive: each chapter title should describe what's in its chapter. Make sure you arrange the titles in the proper order. The order of your chapters should maximise the flow of reading. There should be no forward referencing, so previous chapters should not rely on definitions of concepts which are defined in subsequent chapters.*

*A useful tool in this process is the table of contents. The following is how you use it: (1) open your L<sup>A</sup>T<sub>E</sub>X source file, (2) add a `\tableofcontents`*

*command at the start of your document body, (3) insert your chapter titles with the `\chapter` command, (4) run `latex` twice (why?), and (5) view the table of contents in your browser. Only when you're happy with your titles and their order, should you start writing what is in the chapters. Remember that one of the first things the members of your thesis committee will do is study your table of contents. Better make sure they like it.*

*Note that you may design your chapters in a similar way. Here you start by putting your section titles in the right order. Writing a thesis like this is just like writing a large program in a top-down fashion and filling in the blanks using stepwise refinement.*

### 1.3.3 The Appendix

Some documents have appendices. To indicate the start of the appendix section in your document, you use the `\appendix` command. After that, you use the default commands for starting a sectional unit.

<pre>\appendix \chapter{Proof of Main Theorem} \section{A Useful Lemma}</pre>	<i>LaTeX</i> Input
---	--------------------

## 1.4 Document Management

$\text{\LaTeX}$  input files have a tendency to grow rapidly. If you don't add additional structure then you will lose control over the content even more rapidly. The following three solutions help you stay in control:<sup>3</sup>

**IDE** Use a dedicated  $\text{\LaTeX}$  IDE. A good IDE should let you edit an entire sectional unit as a whole, move it around, and so on. It also should provide a high-level view of the document.

**Folding editor** These are editors which let you define hierarchical folds. A fold works just like a sheet of paper. By folding the fold you hide some of the text. By unfolding the fold or by "entering" a fold you can work on the text that's in the fold.

Folds may be used as follows. At the top level of your  $\text{\LaTeX}$  document you define folds for the top-level sectional units of your document. Within these folds, you define folds for the sub-level sectional units, and so on. By creating folds like this you make the structure of your  $\text{\LaTeX}$  document more explicit, thereby making it easier to maintain your document. For example, re-ordering sectional units is now an easy operation.

**Files**  $\text{\LaTeX}$  has commands which let you include input from other files. By putting the contents of each top-level sectional unit in your  $\text{\LaTeX}$  document in a separate file, you can also make the structure

<sup>3</sup>If you know other solutions then I'd like to learn from you.

**Figure 1.3**

Using the `\includeonly` and `\include` commands. The argument of the `\includeonly` command in the preamble is a list consisting of two file names. These two files are the only files which are included by the `\include` commands

---

```
\includeonly{Abstract.tex,MainResults.tex}
\begin{document}
    \include{Abstract.tex}
    \include{Introduction.tex}
    \include{Notation.tex}
    \include{MainResults.tex}
    \include{Conclusion.tex}
\end{document}
```

---

in the body of the document class. The remaining files are not included, which saves time when latex is run.

in your document more explicit, making it much easier to see the structure.

L<sup>A</sup>T<sub>E</sub>X provides three commands which are related to file inclusion. The first command is `\input`. This command does not provide much flexibility and it is used on its own. Basically, `\input{<file>}` inserts what's in the file `<file>` at the “current” position. The two remaining commands are `\includeonly` and `\include`. These commands provide more flexibility but they are used in tandem. The command `\includeonly{<file list>}` is used in the (document) preamble. It takes one argument, which is a list of the files which may be included further on in the document. To include a file, `<file>`, at a certain position you use the command `\include{<file>}` at that position. If `<file>` is in `<file list>` then it will be included. Otherwise the file will not be included. You can use the command `\include` several times and for several files. The advantage of this conditional file inclusion mechanism is that it saves precious time when working on large documents because non-included files require no latex processing.

Figure 1.3 provides an example with several `\include` commands. The command `\includeonly` at the top of the example tells L<sup>A</sup>T<sub>E</sub>X that only the `\include` statements for the files `Abstract.tex` and `MainResults.tex` should be processed.

## 1.5 Labels and Cross-references

An important aspect of writing a document is *cross-referencing*, i.e. providing references to sectional units, references to tables and figures, and so on. Needless to say, L<sup>A</sup>T<sub>E</sub>X provides support for effective cross-referencing with ease. This section explains the basics for cross-referencing at the document hierarchy level. The mechanism works similar for cross-referencing figures, tables, theorems, and other notions. Note that this section does not study citations. Citations are studied in Section 1.7.

The basic commands for cross-referencing are the commands `\label` and `\ref`.

**Figure 1.4**  
Using `\label` and `\ref`.

---

<code>\chapter{Introduction}</code>	<b>1 Introduction</b>
A short conclusion is presented in Chapter~\ref{TheEnd}.	A short conclusion is presented in Chapter 2.
<code>\chapter{Conclusion}</code>	<b>2 Conclusion</b>
<code>\label{TheEnd}</code>	

---

**Figure 1.5**  
Using `\pageref`.

---

<code>\chapter{Introduction}</code>	<b>1 Introduction</b>
A short conclusion is presented in Chapter~\ref{TheEnd}. The conclusion starts on Page~\pageref{TheEnd}.	A short conclusion is presented in Chapter 2. The conclusion starts on Page 1.
<code>\chapter{Conclusion}</code>	<b>2 Conclusion</b>
<code>\label{TheEnd}</code>	

---

`\label{<label>}`

This defines a logical label, `<label>`, and associates the label with the current environment, i.e. the environment which the `\label` command is in. At the top level, the environment is the current sectional unit. Inside a given theorem environment it is that theorem environment, inside a given figure environment it is that figure environment, and so on. Once defined, the logical label becomes a handle, which you may use to reference “its” environment. The argument of the `\label` command may be any sequence of “normal” symbols. The only restriction is that the sequence be unique. □

`\ref{<label>}`

This command substitutes the number of the environment of the label `<label>`. For example, if `<label>` is the label of the second chapter then `\ref{<label>}` results in ‘2’, if `<label>` is the label of the third section in Chapter 1 then `\ref{<label>}` results in ‘1.3’, and so on. □

Figure 1.4 demonstrates how to use the `\label` and the `\ref` commands. In this example, the tilde symbol (`~`) is used to prevent  $\text{\LaTeX}$  from putting a line-break after the word ‘Section’. In effect it *ties* the words ‘Section’ and the number which is generated by the `\ref` command. Tying text is studied in more detail in Section 2.1.1.

The command `\pageref{<label>}` substitutes the page number “of” the environment of `<label>`. Figure 1.5 demonstrates how to use the `\pageref` command.

If you compile a document which references an undefined label then `latex` will notice this error, complain about it in the form of a warning message, but tacitly ignore the error. Furthermore, it will put two question marks in the output document. The position of the question marks corresponds to the position in the input that references the label. The question marks are typeset in a bold face font: **??**. Even if you fail to notice the warning message this still makes it possible to detect the error.

It should be clear that properly dealing with newly defined or deleted labels requires some form of two-pass system. The first pass detects the label definitions and the second pass inserts the numbers of the labels. When Lamport designed L<sup>A</sup>T<sub>E</sub>X he decided that a two-pass system was too time consuming. Instead he decided to compromise:

- Label definitions are processed by writing them to the auxiliary file for the *next* session.
- Label references are only considered valid when the labels are defined in the auxiliary file of the *current* session.
- If an error occurs, information about labels may not be written to the auxiliary file.

Note that with this mechanism latex cannot know about newly defined labels even if a label is referenced at a position which comes after the definition of the label. This is a common cause of confusion. For example, when latex processes a reference to a label which is not defined in the *current* auxiliary file, it will always output a message warning about new or undefined labels. The warning is output regardless of whether the label is defined in the current input file (as opposed to its being the current auxiliary file). In addition latex will put two question marks where the label is referenced in the text. To the novice user it may seem that they or latex have made an error. However, running latex once more should usually solve the problem and should get rid of the warning message and the question marks.

## 1.6 Controlling the Style of References

The labelling mechanism is elegant and easy to use but you may still run into problems from a document management perspective. For example, in our previous example, we wrote `Chapter~\ref{TheEnd}`, thereby hard-coding the word ‘Chapter’. If for some reason we decided to change ‘Chapter’ to ‘Chap.’ for all references to chapters then we would have to make a change for each reference to a chapter in our source document.

To overcome these problems, and for consistent referencing, it is better to use the `prettyref` package. Using this package adds a bit of intelligence to the cross-referencing mechanism. There are four ingredients to the new cross-referencing mechanism.

1. You introduce classes of elements. Within each class the elements should be referenced in the same way. For example, the class of equations, the class of figures, the class of chapters, the class consisting of sections, subsections, and subsubsections, and so on.
2. You choose a unique prefix for the labels of the classes. For example, ‘eq’ for equations, ‘fig’ for figures, ‘ch’ for chapters, ‘sec’ for sections, subsections, and subsubsections, and so on. Here the prefixes are the first few letters of the class members but this is not required.

**Figure 1.6**

Using the `prettyref` package. The `\newreformat` commands define three classes of labels: ‘ch’, ‘sec’, and ‘fig’. The command `\newreformat` has two arguments. The first argument determines the class and the second determines how the command typesets labels from that class. For example, labels starting with ‘ch:’ are typeset as ‘Chapter’ followed by the number of the label.

---

```
\usepackage{prettyref}
\newreformat{ch}{Chapter~\ref{#1}}
\newreformat{sec}{Section~\ref{#1}}
\newreformat{fig}{Figure~\ref{#1}}
\begin{document}
  \chapter{Introduction}
  In \prettyref{ch:Main@Results}
    we present the main results.
  \chapter{Main Results}
  \label{ch:Main@Results}
  ...
\end{document}
```

---

3. You use the `\newreformat` command to specify how each class should be referenced. You do this by telling the command about the unique prefix of the class and the text that should be used for the reference. For example, `\newreformat{ch}{Chapter~\ref{#1}}` states that the unique label prefix ‘ch’ is for a class of elements that have references of the form ‘Chapter~\ref{#1}’, where ‘#1’ is the logical label of the element (including the prefix). As another example, `\newreformat{id}{\ref{#1}}` gives you the same reference you get if you apply `\ref` to the label.
4. You use `\prettyref` instead of `\ref`. This time the labels are of the form ‘(prefix): (label)’. For example, `\prettyref{fig:fractal}`, `\prettyref{ch:Introduction}`, and so on.

Changing the style of the cross-references of a class now only requires changing one call to `\newreformat`. Clearly, this is a better cross-referencing mechanism. Since `prettyref` is a package, it should be included in the preamble of your  $\text{\LaTeX}$  document. Figure 1.6 provides a complete example of the `prettyref` mechanism.

## 1.7 The Bibliography

### 1.7.1 Basic Usage

Most scholarly computer science works have citations and a bibliography. The purpose of the bibliography is to provide details of the works that are cited in the text. The purpose of the citation is to acknowledge the cited work and to inform your readers how to find the work. In computer science the bibliography is usually at the end of the work. However, in some scientific communities it is common practice to have a bibliography at the end of each chapter in a book. Other communities (e.g., history) use *note systems*. These systems use numbers in the text which refer to footnotes or to notes at the end of the chapter, paper, or book.



**Figure 1.7**  
A minimal bibliography.

---

[Lamport, 1994] L. Lamport. L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System. Addison-Wesley, 1994.  
 [Knuth, 1990] D. E. Knuth. The T<sub>E</sub>Xbook. Addison-Wesley, 1990. The source of this book is freely available from <http://www.ctan.org/tex-archive/systems/knuth/tex/>.

---

The bibliography entries are listed as ‘⟨citation label⟩ ⟨bibliography content⟩’. The ⟨citation label⟩ of a given work is used when the work is cited in the text. The ⟨bibliography content⟩ lists the relevant information about the work. Figure 1.7 presents an example of two entries in a bibliography. For this example, the citation labels are typeset in boldface font inside square brackets.

Even within a single work there may be different styles of citations. *Parenthetical citations* are usually formed by putting one or several citation labels inside square brackets or parentheses. However, there are also other forms of citations which are derived from the information in the citation label.

Within one single bibliography the bibliography content of different kinds of works may differ. For example, entries of journal articles have page numbers but book entries do not.

Bibliographies in different works may also differ. They may have different kinds of (citation) labels and different information in the bibliography content. The order of presentation of the entries in the bibliographies may also be different. For example, entries may be listed alphabetically, in the order of first citation in the text, ...

In L<sup>A</sup>T<sub>E</sub>X the style of the bibliography and labels is configurable. Labels may appear as:

**Numbers** This style results in citations which appear as ‘[⟨number⟩]’ in the text.

**Names and years** This style results in citations which appear as ‘[⟨name⟩, ⟨year⟩]’ in the text.

...

**Labels as Numbers** Labels as numbers are very compact. They don’t disrupt the “flow of reading”: they’re easy to skip. Unfortunately, labels as numbers are not very informative. You have to look up which work corresponds to the number in the bibliography. This may be annoying because it hinders the reading process. What is worse, labels as numbers lack so much information content that you may have to look up the number several times. However, hyperlinks in electronic documents somewhat reduce this problem.

**Labels as Names and Years** Labels as names and year are longer than labels as numbers. They are more disruptive to the reading process: they

**Figure 1.8**  
The `\cite` command.

---

The `\LaTeX{}` package was created by Leslie Lamport%  
`~\cite{Lamport:94}`  
 on top of Donald Knuth's  
`\TeX{}` program%  
`~\cite{Knuth:1990}`.

---

The `\LaTeX` package was created by Leslie Lamport [Lamport, 1994] on top of Donald Knuth's `\TeX` program [Knuth, 1990].

---

are more difficult to “skip”. However, labels as names and years are more informative. If you’re familiar with the literature then usually there’s no need to go to the bibliography to look up the label. Even if you have to look up which work corresponds to a label you will probably remember it the next time you see the label. Compared to labels as numbers this is a great advantage.

Traditionally, labels for citations appeared as numbers in the text. The main reason for doing this was probably to keep the printing costs low. Nowadays, printing costs are not always relevant.<sup>4</sup> For example, paper is not as expensive as before. Also many documents are distributed electronically. Some journals and universities require specific bibliography style/format.

The `\bibliographystyle` command tells `\LaTeX` which style to use for the bibliography. The bibliography style called `(style)`, is defined in the file `(style).bst`. The following demonstrates how you use the `\bibliographystyle` command for selecting the bibliography style called `'named'`, which is the style that is used in this book. Though this is not required, it is arguably a good idea to put the `\bibliographystyle` command in the preamble of your document. The bibliography style `named` requires the additional package `named`, which explains why the additional command `\usepackage{named}` is used in the example.

```
\bibliographystyle{named}
\usepackage{named}
```

*LaTeX* Input

The following are a few commonly used bibliography styles. This list is based on [Lamport, 1994, pages 70–71].

**plain** Entries are sorted alphabetically. Labels appear as numbers in the text.

**alpha** Entries are sorted alphabetically. Labels are formed from surnames and year of publication (e.g., Knut66).

**abbrv** Entries are very compact and sorted alphabetically. Labels appear as numbers in the text.

Citing a work in `\LaTeX` is similar to referencing a section. Both mechanisms use logical labels. For referencing you use the `\ref` command but

---

<sup>4</sup>But we should think about the environmental effects of using more paper than necessary.

**Figure 1.9** Using `\cite` with an optional argument.

---

More information about the bibliography database may be found in%  
`\cite[Appendix~B]{Lamport:94}`.  


---

More information about the bibliography database may be found in [Lamport, 1994, Appendix B].

for citations you use the `\cite` command. The argument of the `\cite` command is the logical label of the work you cite.

Figure 1.8 provides an example. The example involves two logical labels: ‘Lamport:94’ and ‘Knuth:1990’. Each of them is associated with a work in the bibliography. The first label is the logical label of a book by Lamport; the second that of a book by Knuth. As it happens the names of the labels are similar to the resulting citation labels but this is not required. The command ‘`\cite{Lamport:94}`’ results in the citation ‘[Lamport, 1994]’. Here ‘Lamport, 1994’ is the citation label of Lamport’s book in the bibliography. This label is automatically generated by the BibT<sub>E</sub>X program. This is explained further on.

The reason for putting an empty group (the two braces) after the `\LaTeX` and `\TeX` commands is technical. The following explains this in more detail. In E<sub>T</sub><sub>X</sub> a group is treated as a word. However, L<sup>A</sup>T<sub>E</sub>X ignores spaces after most commands, including `\TeX` and `\LaTeX`. Without the empty groups, there would not have been proper inter-word spacing between ‘L<sup>A</sup>T<sub>E</sub>X’ and ‘package’ and between ‘T<sub>E</sub>X’ and ‘program’ in the final output. However, adding the empty groups after the commands results in the proper inter-word spacing.

It may not be immediately obvious, but in the example of Figure 1.8 the text ‘Lamport’ on Line 2 in the input is still tied to the command ‘`\cite{Lamport:94}`’ which is on the following line. The reason is that the comment following the text ‘Lamport’ makes L<sup>A</sup>T<sub>E</sub>X ignore all input until the next non-space character on the next line.

You can also cite parts of a work, for example a chapter or a figure. This is done by passing an optional argument to the `\cite` command which specifies the part. The example in Figure 1.9 demonstrates how you do this.

The following commands are also related to the bibliography.

`\refname`

This results in the name of the bibliography section. In the article class, the command `\refname` is initially defined as ‘References’. □

`\renewcommand{\refname}{\langle other name \rangle}`

This redefines the command `\refname` to `\langle other name \rangle`. The `\renewcommand` may also be used to redefine other existing commands. It is explained in Chapter 10. □

`\nocite{\langle list \rangle}`

This produces no text but writes the entries in the comma-separated list `\langle list \rangle` to the bibliography file. If you use this command, then you should consider making it very clear which works in the bibliography

are not cited in the text. For example, some readers may be interested in a discussion of these uncited works, why they are relevant, and so on. They may get very frustrated if they can't find citations to these works in your text. □

### 1.7.2 The `bibtex` Program

Since bibliographies are important and since it's easy to get them wrong, some of the work related to the creation of the bibliography has been automated. This is done by `BIBTEX`. The `BIBTEX` tool requires an external human-readable bibliography database. The database may be viewed as a collection of records. Each record defines a work that may be listed in the bibliography. The record defines the title of the work, the author(s) of the work, the year of publication, and so on. The record also defines the logical label of the work. This is the label you use when you `\cite` the work.

The advantage of using `BIBTEX` is that you provide the information of the entries in the bibliography and that `BIBTEX` generates the text for the bibliography. This guarantees consistency and ease of mind. For example, changing the style of the bibliography is a piece of cake. Furthermore, the `BIBTEX` database is reusable and you may find `BIBTEX` descriptions of many scholarly works on the web. A good starting point is <http://citeseer.ist.psu.edu/>.

Generating the bibliography with `BIBTEX` is a multi-stage process, which requires an external program called `bibtex`. The `bibtex` program is to `BIBTEX` what the `latex` program is to `TEX`. The following explains the process.

1. You specify the bibliography with the `\bibliography` command. The command takes one argument which is the basename of the `BIBTEX` database, so if you use `\bibliography{<db>}` then your database is `<db>.bib`.
2. You `\cite` works in your `TEX` program. Your logical labels should be defined by some `BIBTEX` record.
3. You run `latex`. This writes the logical labels to an auxiliary file.
4. You run `bibtex` as follows:

```
$ bibtex <document>
```

Unix Usage

Here `<document>` is the basename of your top-level `TEX` document. The `bibtex` program will pick up the logical labels from the auxiliary file, look up the corresponding records in the `BIBTEX` database, and generate the code for `TEX`'s bibliography. A common mistake of `bibtex` users is that they add the `'.tex'` extension to the basename of the `TEX` source file. It is not clear why this is not allowed.

5. You run `latex` twice (why?) and Bob's your uncle.

It is important to understand that you (may) have to run `bibtex` when (1) new citation labels are added, when (2) existing citation labels are

**Figure 1.10**  
Including a bibliography.

---

```
\documentclass[11pt]{article}
% Use bibliography style named.
% Requires the file named.bst.
\bibliographystyle{named}
% Requires the package named.sty.
\usepackage{named}
\begin{document}
  % Put in a citation.
  This cites~\cite{Knuth:1990}.
  % Put the reference section here.
  % It is in the file db.bib.
  \bibliography{db}
\end{document}
```

---

**Figure 1.11**  
Some BIB<sub>T</sub>E<sub>X</sub> entries.

---

```
@Book{Lamport:94,
  author    = {Lamport, Leslie},
  title     = {\LaTeX: A Document Preparation System},
  year      = {1994},
  isbn      = {0-021-52983-1},
  publisher = {Addison-Wesley},
}

@Book{Strunk:White:1979,
  author    = {Strunk, W. and
              White, E.~B.},
  title     = {The Elements of Style},
  publisher = {Macmillan Publishing},
  year      = {1979},
}
```

---

removed, and when (3) you change the BIB<sub>T</sub>E<sub>X</sub> records of works in your bibliography. Each time you run `bibtex` should be followed by two more `latex` runs.

Figure 1.10 provides an example. The L<sup>A</sup>T<sub>E</sub>X source in this example depends on a BIB<sub>T</sub>E<sub>X</sub> file called ‘db.bib’.

Figure 1.11 presents an example of two entries in a BIB<sub>T</sub>E<sub>X</sub> database file. The example associates the logical label ‘Lamport:94’ with Lamport’s L<sup>A</sup>T<sub>E</sub>X book and the logical label ‘Strunk:White:1979’ with the book about elements of style. The author, title, year of publication, International Standard Book Number (ISBN), and the publisher of the book are also specified in the entries. Depending on the style which is used to generate the bibliography, some of this information may or may not appear in the references. Notice that the author names are specified by first providing the surname and then providing the first name(s). The surname and first name(s) are separated with a comma. The second entry in the example shows that multiple authors are separated with the

keyword ‘and’.

Now that you know how to use the `bibtex` program, let’s see what you can put in the `BIBTEX` database. The following list is not exhaustive. For a more accurate list you may wish to read [Fenn, 2006; Lamport, 1994]. The following is based on [Lamport, 1994, Appendix B].

**@Article:** An article from a journal or magazine.

**Required entries** author, title, journal, and year.

**Optional entries** volume, number, pages, month, and note.

**@Book:** A book with an explicit publisher.

**Required entries** author or editor, title, publisher, and year.

**Optional entries** volume, number, series, ....

**@InProceedings:** A paper in a conference proceedings.

**Required entries** author, title, booktitle, publisher, and year.

**Optional entries** pages, editor, volume, number, series, ....

**@Proceedings:** The proceedings of a conference.

**Required entries** title and year.

**Optional entries** editor, volume, number, series, organisation,  
....

**@MastersThesis:** A Master’s thesis.

**Required entries** author, title, school, and year.

**Optional entries** type, address, month, and note.

**@PhDThesis:** A Ph.D. thesis.

**Required entries** author, title, school, and year.

**Optional entries** type, address, month, and note.

....

An impressive list of `BIBTEX` style examples may be found at <http://www.cs.stir.ac.uk/~kjt/software/latex/showbst.html>.

### 1.7.3 The `natbib` Package

There are several problems with the basic `ETEX` citation mechanism. The `natbib` package overcomes some of them. It also provides a more flexible citation mechanism.

**Figure 1.12**

The `\citet` and `\citep` commands.

<code>\citet{Knuth:1990}</code>	Knuth (1990) describes T <sub>E</sub> X. The ultimate guide to T <sub>E</sub> X is (Knuth, 1990).
describes \TeX.	
The ultimate guide to \TeX{}	
is~\citep{Knuth:1990}.	

**Figure 1.13**

Using the `\citeauthor` and `\citeyear` commands.

<code>\citeauthor{Knuth:1990}</code>	Knuth wrote (Knuth, 1990) in 1990.
wrote~\cite{Knuth:1990}	
in~\citeyear{Knuth:1990}.	

- The `natbib` package distinguishes between *parenthetical* and *textual* citations. A parenthetical citation is similar to the default L<sup>A</sup>T<sub>E</sub>X citation. They are mainly used to provide a reference to the work: leaving them out should leave the grammar of the sentence intact. With `natbib` you get parenthetical citations with the `\citep` command. Textual citations play an active rôle in the sentence: leaving them out should result in grammatical errors. With `natbib` you get such citations with the `\citet` command. Figure 1.12 demonstrates how these commands work.
- The package also provides the command `\citeauthor` and `\citeyear`. The first command gives you the author(s) and the second the year of a citation. Figure 1.13 shows how to use these commands.
- An important improvement is that `natbib` lets you capitalise “von” parts in surnames. To achieve this you use similar commands as before. However, this time the relevant commands start with an upper case letter. The following demonstrates how this works.

```
\Citeauthor{Beethoven:ninth}
is most famous for his Ninth Symphony%
~\Citet{Beethoven:ninth}.
Pesonally, I prefer his Sixth Symphony%
~\Citet{Beethoven:sixth}.
```

- Finally, `natbib` lets you specify the style of the labels which are used for the citation in the text. By default `natbib` uses parentheses for parenthetical citations.
- ....

You can get information about the `natbib` package by executing the following command.

```
$ texdoc natbib
```

Getting help for other packages and classes works similarly. You may download the `natbib` package from the Comprehensive T<sub>E</sub>X Archive Network (CTAN), which may be found at <http://www.ctan.org>. If you are looking for other classes and packages then CTAN is also the place to be.

### 1.7.4 Multiple Bibliographies

This section explains how you create documents with more than one bibliography. The `multibbl`, `multibib`, and `bibtopic` packages support multiple bibliographies. The following explains how you use the `multibbl` package.

Suppose you want separate bibliographies for books and articles (other bibliographies are created analogously). The following explains what you do on the  $\text{\LaTeX}$  side.

1. You include the `multibbl` package. This is done in the usual way.
2. The `multibbl` package requires a unique name for each bibliography. You specify these names with the `\newbibliography` command. Let's them `books` and `articles`.

```
\newbibliography{books}
\newbibliography{articles}
```

*$\text{\LaTeX}$  Input*

3. Using the `\bibliographystyle` command — it is redefined by the `multibbl` package — you define a bibliography style for the bibliographies. The following uses the same style for the bibliographies but this is not required.

```
\bibliographystyle{books}{named}
\bibliographystyle{articles}{named}
```

*$\text{\LaTeX}$  Input*

4. You put citations in your document with the redefined `\cite` command.

```
The ultimate guide to \TeX{} is%
~\cite{books}{Knuth:1990}.
```

*$\text{\LaTeX}$  Input*

This time `\cite` takes two arguments. The first argument is the name of the bibliography. The second argument is the usual citation label. Optional arguments are handled as per usual.

```
\cite[Chapter~2]{articles}{Fenn:2006}
describes how to use \BibTeX.
```

*$\text{\LaTeX}$  Input*

5. To create the bibliography, you use the redefined `\bibliography` command.

```
\bibliography{books}{db}{Books about \LaTeX}
\bibliography{articles}{db}{Articles about \LaTeX}
```

*$\text{\LaTeX}$  Input*

Compared to the original `\bibliography` command, the new commands takes two more arguments. As before `\bibliography{<name>}{<db>}{<title>}` inserts a bibliography. This time, the first argument is the name of the bibliography. The second argument is the basename of the  $\text{\BibTeX}$  database file. The previous example, uses the same  $\text{\BibTeX}$  database for the bibliographies but this is not a requirement. The last argument is the title of the bibliography. It also appears in the table of contents.

6. It is usually useful to add an extra line to the table of contents that indicates the start of the bibliographies. The following example



shows how you add the word ‘Bibliographies’ to the table of contents with the starred version of the `\part` command. (Remember that the starred version does not result in a number.)

```
\part*{Bibliographies}
\bibliography{books}{db}{Books about \LaTeX}
\bibliography{articles}{db}{Articles about \LaTeX}
```

L<sup>A</sup>T<sub>E</sub>X Input

We’re done with the work at the L<sup>A</sup>T<sub>E</sub>X level. This time we use BIB<sub>T</sub>E<sub>X</sub> and apply it to the names of the bibliographies.

```
$ bibtex books
$ bibtex articles
```

Unix Usage

### 1.7.5 Bibliographies at End of Chapter

Some documents require a bibliography at the end of each chapter. Should you require them then the packages `chapterbib` and `bibunits` may be useful.

To Do

## 1.8 Reference Lists

### 1.8.1 Table of Contents and Lists of Things

This section explains how to include a table of contents and *reference lists* in your document. Here a reference list is a list which tells you where (in the document) you may find certain things. Common examples are a list of figures, a list of tables, and so on. L<sup>A</sup>T<sub>E</sub>X also lets you define other reference lists. The following example, which should be easy enough to understand, shows how you include a table of contents, and lists of figures and tables.

```
\begin{document}
  \maketitle
  \include{Abstract.tex}
  \clearpage
  \tableofcontents
  \listoffigures
  \listoftables
  :
\end{document}
```

L<sup>A</sup>T<sub>E</sub>X Input

In the example, the `\clearpage` command inserts a pagebreak after the first `\include` command. As a side-effect it also forces any figures and tables that have so far appeared in the input to be printed. There is also a command called `\cleardoublepage`, which works similarly. However, in a two-sided printing style, it also makes the next page a right-hand side (odd-numbered) page, producing a blank page if necessary.

**Table 1.1**

Depth values of sectional unit commands. The first column in the table lists the sectional unit commands. For each command, the corresponding value of the `\tocdepth` counter is listed in the second column. That of the `\secnumdepth` counter value is listed in the last column.

Sectional Unit Command	<code>\tocdepth</code>	<code>\secnumdepth</code>
<code>\part</code>	-1	1
<code>\chapter</code>	0	2
<code>\section</code>	1	3
<code>\subsection</code>	2	4
<code>\subsubsection</code>	3	5
<code>\paragraph</code>	4	6
<code>\subparagraph</code>	5	7

### 1.8.2 Controlling the Table of Contents

The *counter* `\tocdepth` (counters are discussed in Chapter 12) gives some control over what is listed in the table of contents. The value of the counter controls the depth of last sectional level that is listed in the table of contents. The value 0 represents the highest sectional unit, 1 the next sectional unit, and so on.

By setting the value of `\tocdepth` to `<depth>` you limit the depths of the sectional units that are listed in the table of contents from 0 to `<depth>`. For example, if you're using the `book` class, then using 0 for `<depth>` will allow parts and chapters in the table of contents, but not sections. As another example, if you're using the `article` class, then using 2 for `<depth>` will only list sections, subsections, and subsubsections in the table of contents. You set the counter `\tocdepth` to `<depth>` with the command `'\setcounter{\tocdepth}{<depth>}'`.

### 1.8.3 Controlling the Sectional Unit Numbering

The counter `\secnumdepth` is related to the counter `\tocdepth`. Its value determines the depth of the sectional units that are numbered. So by setting the counter `\secnumdepth` to 3, you tell  $\text{\LaTeX}$  to number parts, chapter, and sections, and tell it to stop numbering subsections and less significant sectional units.

Table 1.1 lists the sectional unit commands and the corresponding numbers for the counters `\tocdepth` and `\secnumdepth`.

### 1.8.4 Indexes and Glossaries

If you are writing a book or a thesis, you probably want to include an index or glossary of some kind. Getting it to work may take a while. The remainder of this section explains how to create an index. The mechanism for glossaries is similar.

Unfortunately,  $\text{\LaTeX}$ 's default index mechanism only allows you to have one single index. The `multind` package lets you create several index lists. The package works as follows:

1. You associate each index with a file name. You do this by passing

passing the basename of the file to the command `\makeindex`.

```
\makeindex{programs}
\makeindex{authors}
```

*L<sup>A</sup>T<sub>E</sub>X* Input

2. You insert the indexes with the `\printindex` command.

```
\printindex{programs}{Index of Programs}
\printindex{authors}{Index of Authors}
```

*L<sup>A</sup>T<sub>E</sub>X* Input

The first argument of `\printindex` is the name of the corresponding index. The second name is the title of the index. The title also appears in the table of contents.

3. You define the index entries. You use the `\index` command to define what is in the indexes. The following is a simple example which creates an entry ‘TeX’ in the index for the programs.

```
Knuth\index{authors}{Knuth}
    is the author of \TeX\index{programs}{TeX}.
```

*L<sup>A</sup>T<sub>E</sub>X* Input

Behind the scenes the `\index` command writes information to the auxiliary files `authors.idx` and `programs.idx`. In the following step we shall use the `makeindex` program to turn it into files which can be included in our final document.

4. You process the `.idx` files with the program `makeindex`. This is similar to using `bibtex` for generating the bibliography. The following demonstrates how to use the program.

```
$ makeindex authors
$ makeindex programs
```

Unix Session

The remainder of this section explains how you create more complex indexes with the `multind` package. The `multind` package redefines the `\index` command. The redefined command takes one more argument. The first argument of the redefined command determines the name of an auxiliary file which is used to construct the index. The last argument of the redefined command describes the index entry. The following is based on [Lamport, 1994, Appendix A.2].

```
\index{<name>}{<entry>}
```

This creates an index entry for `<entry>`. The entry also lists the page number. □

```
\index{<name>}{<entry>}{<subentry>}
```

This creates a subentry `<subentry>` for the index entry `<entry>`. It also lists the page number. □

```
\index{<name>}{<entry>}{<subentry>}{<subsubentry>}
```

This creates a sub-subentry `<subsubentry>` for subentry `<subentry>` for the index entry `<entry>`. It also lists the page number. □

```
\index{<name>}{<entry>|see{<other entry>}}
```

This creates a cross-reference to `<other entry>` in the entry for `<entry>`. This does not result in a page number for `<entry>`. □

```
\index{<name>}{<entry for sorting>@{<entry for printing>}}
```

This results in an entry for `<entry for printing>` in the index list. The

Table 1.2

Creating indexes with the `\index` command. The table to the left lists the last argument of the `\index` command and the corresponding page in the output document. So if you're using the unmodified `\index` command then the second column in the table corresponds to the first argument of `\index`. However, if you're using the `multind` package then the column corresponds to the command's second argument. The output to the right is the resulting index.

Page	Last argument of the <code>\index</code> command
1	lecture notes
2	programs
4	lard
2	latex@ <code>\LaTeX</code>
3	lambda@ <code>\lambda</code>
5	sausages!boerewors
6	sausages!salami
2	programs!latex
6	programs!bibtex
2	index ( <code></code>
6	index ) <code></code>
8	salami  <code>see{sausages}</code>
8	boerewors  <code>see{sausages}</code>
8	boereworst (Dutch)  <code>see{boerewors}</code>

Index	<i>LaTeX</i> Output
boerewors, <i>see</i> sausages	
boereworst (Dutch), <i>see</i> boerewors	
index, 2–6	
$\lambda$ , 3	
lard, 4	
<code>\LaTeX</code> , 2	
lecture notes, 1	
programs, 2	
bibtex, 6	
latex, 2	
salami, <i>see</i> sausages	
sausages	
boerewors, 5	
salami, 6	

position in the index is determined by `<entry for sorting>`. This is useful, for example, if `<entry for printing>` contains mixed upper- and lowercase letters or if it contains other characters. For example,

- `\index{<name>}{twenty@20};`
- `\index{<name>}{twenty@xx};`
- `\index{<name>}{beta@ $\beta$ };` or
- `\index{<name>}{command@\textbackslash command}.` □

There is one more construct which is useful for topics which cover a page range. To create an entry for topic `<topic>` for a page range, you start the range with the command `\index{<name>}{<topic>|(}` and you close the range with the command `\index{<name>}{<topic>|)}`.

Table 1.2 presents an example of the `\index` command. The left of the figure depicts the last argument of the `\index` commands and the current page of the `LaTeX` output. It is assumed that the first argument of the `\index` command is the same. The right of the figure depicts the resulting index. Notice that the entries for 'programs' and 'sausages' both have subentries. However, the entry for 'programs' has a page number whereas the entry for 'sausages' does not. To understand this difference you need to know that the page number for top-level entries is generated by commands of the form `\index{<name>}{<entry>}`. Since there is such a command for 'sausages' but not for 'programs' this explains the difference. More information about the `\index` command may be found in [Lamport, 1994, Appendix A].

## 1.9 Class Files

As explained before, each top-level L<sup>A</sup>T<sub>E</sub>X document corresponds to a document class. The document class is determined by the required argument of `\documentclass` command in your L<sup>A</sup>T<sub>E</sub>X document.

```
\documentclass{<document class name>} LATEX Input
```

Each document class is defined in a class file. Class files define the general rules for typesetting the document. It is recalled that you may pass options to classes. This is done by putting the options inside the square brackets following the command `\documentclass`. If you have multiple options then you separate them with commas.

The extension of class files is `.cls`. The following are some standard class files.

**article** The basic article style. The top-level sectional unit of this class is the section.

**book** The basic book style. The top-level sectional unit of this class is the chapter. The `book` class also provides the commands for indicating the start of the front, main, and back matter.

**report** The basic report style. The top-level sectional unit of this class is the chapter.

**letter** The basic style for letters. This class has no sectional units. The letter is written inside a `letter` environment, which takes one required argument which is the address of the person you are writing the letter to. In addition there are commands for specifying the address of the writer, the signature, the opening and closing lines, the “carbon copy” list, and enclosures and postscriptum. More detailed information about the `letter` class may be found in [Lamport, 1994, Page 84–86] and on <http://en.wikibooks.org/wiki/LaTeX/Letters>. Figure 1.14 presents a minimal example of a letter.

The following options are typically available for the previous class files.

**11pt** Uses an 11 point font size instead of the 10 point size, which is the default.

**12pt** Uses a 12 point font size.

**twoside** Output a document which is printed on both sides of the paper.

**twocolumn** Output a document which has two columns.

**draft** Used for draft versions. This option makes L<sup>A</sup>T<sub>E</sub>X indicate hyphenation and justification problems by putting a little square in the margin of the problem line.

**final** Used for the final version.

**Figure 1.14**  
Minimal letter.

---

```

\documentclass{letter}
% Sender details.
\signature{T.~Dee}
\address{Give Cash\\Dublin}

\begin{document}
% Addressee. A double backslash generates a newline.
\begin{letter}{Get Cash\\Cork}
  \opening{Dear Sir/Madam:}

  Please make a cash donation to our party.

  We look forward to the money.

  \closing{Yours Faithfully,}
  \ps{P.S.\ Send it now.}
  \encl{Empty brown envelope.}
  \cc{Paddy.}
\end{letter}
\end{document}

```

---

## 1.10 Packages

Document classes are fairly minimal. Usually, you need some additional commands for doing your day-to-day document preparation. This is where *packages* (originally called *style files*) come into play. Packages have the following purpose.

**Provide commands** Define or redefine a useful command. Usually, this adds some extra functionality.

**Change settings** Tweak some default document settings. Usually, this affects the layout.

The extension of packages is `.sty`. You include the package which is defined in the file `\style.sty` as follows.

```
\usepackage{\style}
```

*LaTeX* Input

Some packages accept options. You pass them to the package using the same mechanism as with class files. Multiple options are separated using commas. The following shows how to pass the options ‘first’ and ‘second’ to the fictional package `counter`.

```
\usepackage[first,second]{counter}
```

*LaTeX* Input

To find out about the options you have to read the documentation. Remember that `texdoc` (see Page 27) helps you locate and read the documentation.

## 1.11 Useful Classes and Packages

There are hundreds, if not thousands, of existing classes and packages (packages are also known as style files). The following are some useful classes and packages.<sup>5</sup>

**listings** Including program listings [Heinz and Moses, 2007].

**url** Typesetting URLs [Arseneau, 2010].

**prettyref** Consistent typesetting cross-references [Ruland, 2007].

**amsmath** Typesetting tools from the American Mathematical Society (AMS) [American Mathematical Society, 2002].

**fourier** Sets the text font to *Utopia Regular* and the math font to *Fourier* [Bovani, 2005].

**graphicx** Including graphics [D. P. Carlisle, 2003].

**coverpage** User-defined coverpages [Mühlich, 2006].

**fancyhdr** User-defined headers and footers [Oostrum, 2004].

**lastpage** Defines a command for getting the last page number. This is especially useful for  $M/N$  page numbers [Goldberg, 2010].

**memoir** This class provides support for writing books. The class comes with lots and lots of options for finetuning the typesetting [Wilson, 2010].

**keyval**

**xkeyval** Supports parsing of `key=val` lists [D. Carlisle, 1999b; Adriaens, 2008].

**classicthesis** Nice package for thesis [Miede, 2010].

**mathtools** More precise typesetting of math [Høgholm, 2010].

## 1.12 Errors and Troubleshooting

Errors and `texmf.cnf`. **To Do.**

---

<sup>5</sup>If you have other favourites then please let me know.





# **Part II**

## **Basic Typesetting**



# CHAPTER 2

## Running Text

THIS CHAPTER EXPLAINS everything you’ve always wanted to know about writing text, aligning it, and changing text appearance. It is recalled from Chapter 1 that  $\text{\LaTeX}$  is implemented on top of  $\text{\TeX}$ , which is a rewriting machine which turns token streams into token streams. Some of the character tokens in the input stream have a special meaning to  $\text{\TeX}$  (and  $\text{\LaTeX}$ ). These tokens are studied in Section 2.1.

Having studied the special tokens we are ready to do some fancy typesetting. We start with some sections about diacritics, ligatures, dashes, emphasis, footnotes and marginal notes, quotes and quotations. If you’re not familiar with some of these notions then don’t worry because they are explained further on in this chapter. The next sections are about changing the size of the text, changing the type style of the text, and phantom (invisible) text. The remaining sections cover the most important text alignment techniques and language related issues.

### 2.1 Special Characters

This section studies ten characters which have a special meaning to  $\text{\TeX}$ . When  $\text{\TeX}$  sees these characters as tokens in the input stream, then it usually does not typeset them but, instead, changes state. The remainder of this section briefly explains the purpose of the tokens and how you typeset them as characters in the output.

Table 2.1 depicts the tokens, their meaning, and the command to typeset them. We have already studied the start-of-comment token (%) and the backslash (\), which starts control sequences. Typsetting a backslash is done with the commands `\textbackslash` and `\backslash`. The latter command is only used when specifying mathematical formulae. It is described in Chapter 8. The command argument reference token is described in Chapter 10. The alignment tab (&) is described in Section 2.14.3. This token usually indicates vertical alignment positions in array-like structures consisting of rows and columns. The math mode switch token (\$), the subscript token (\_), and the superscript token (^) are described in Chapter 8. The three remaining tokens are described in the remainder of this section.

Table 2.1

The characters in the first column have a special meaning to  $\text{\LaTeX}$ . The purpose of the characters is listed in the column ‘Purpose’. The last column lists the command which produces the character. The command `\textbackslash` is used when typesetting normal text. The command `\backslash` is used when typesetting mathematics.

Character	Purpose	Command
#	Formal parameter	<code>\#</code>
\$	Math mode switch	<code>\\$</code>
%	Start of comment	<code>\%</code>
&	Alignment tab	<code>\&amp;</code>
~	Text tie token	<code>\textasciitilde</code>
_	Math subscript	<code>\_</code>
^	Math superscript	<code>\textasciicircum</code>
{	Start of group	<code>\{</code>
}	End of group	<code>\}</code>
\	Start of command	<code>\textbackslash</code> or <code>\backslash</code>

### 2.1.1 Tying Text

Remember that  $\text{\LaTeX}$  is a large rewriting machine which repeatedly turns token sequences into token sequences. At some stage it turns a token sequence into lines. This is where  $\text{\LaTeX}$  ( $\text{\TeX}$  really) determines the line breaks. The tilde token (`~`) defines an inter-word space which cannot be turned into a line break. As such it may be viewed as an operator which ties words.

The following example demonstrates two important applications of the tilde operator: it prevents unpleasant linebreaks in references and citations.

$\text{\LaTeX}$  Usage

```
... Figure%
~\ref{fig:list@format}
depicts the format of a list.
It is a reproduction of%
~\cite[Figure~6.3]{Lamport:94}.
```

It is usually not too difficult to decide where to use the tie operator. The following are some concrete examples, which are taken from [Knuth, 1990, Chapter 14].

- References to named parts of a document:
  - Chapter~12,
  - Theorem~1.5,
  - Lemmas 5 and~6,
  - ....
- Between a person’s forenames and between multiple surnames:
  - Donald~E. Knuth,
  - Luis~I.~Trabb~Pardo,
  - Bartel~Leendert van~der~Waarden,
  - Charles~XII,
  - ....

- Between math symbols in apposition with nouns:

- `dimension~$d$`,
- `string~$s$ of length~$l$`,
- ....

Here the construct `$\langle\math\rangle$` is used to typeset  $\langle\math\rangle$  as an in-line mathematical expression.

- Between symbols in series:

- `1,~2, or~3`.

- When a symbol is a tightly bound object of a preposition:

- `from 0 to~1`,
- `increase $z$ by~1`,
- ....

- When mathematical phrases are rendered in words:

- `equals~$n$`,
- `less than~$\epsilon$`,
- `modulo~$2$`,
- `for large~$n$`,
- ....

- When cases are being enumerated within a paragraph:

- `(b)~Show that function $f(x)$ is (1)~continuous;`  
`(2)~bounded.`

## 2.1.2 Grouping

Grouping is another common technique in  $\text{T}_{\text{E}}\text{X}$  and  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ . The left brace token (`{`) starts a group. The right brace token (`}`) closes it. Grouping has two purposes: The first purpose of grouping is that it turns a number of things into one compound thing. This may be needed, for example, if you want to pass several words as the single argument to a command which typesets its sole argument in bold face text. The following demonstrates the point.

A bold <code>\textbf{word}</code> and a bold <code>\textbf{letter}</code> .	A bold word and a bold letter.
--	--------------------------------

The second purpose of grouping is that it lets you change certain settings and keep the changes local to the group. The following demonstrates how this may be used to make a local change to how the text is typeset inside the group.

Normal text here. {% Start a group. <code>\bfseries</code> % Now we have bold text. Bold paragraphs in here. }% Close the group. Back to normal text again.	Normal text here. <b>Bold paragraphs in here.</b> Back to normal text again.
---	--

Table 2.2  
Common diacritics.

Output	Command	Name
ò	\`{o}	Acute accent
ó	\'{o}	Grave accent
ô	\^{o}	Circumflex (hat)
õ	\~{o}	Tilde (squiggle)
ö	\" {o}	Umlaut or dieresis
ċ	\. {c}	Dot accent
š	\v{s}	Háček (caron or check)
ö	\u{o}	Breve accent
ō	\={o}	Macron (bar)
ő	\H{o}	Long Hungarian umlaut
ôo	\t{oo}	Tie-after accent
ç	\c{s}	Cedilla accent
ḡ	\d{o}	Dot-under accent
ḡ	\b{o}	Bar-under accent

Inside the group you may have several paragraphs. The advantage of the declaration `\bfseries` is that it lets you define how the text is typeset for the remainder of the group, which may have several paragraphs. This is an advantage to the `\textbf` command, which does not allow paragraph-breaks in its argument.

There is also a low-level T<sub>E</sub>X operator pair for creating groups. It works just as the braces. A group is started with `\begingroup` and ended with `\endgroup`. These tokens may be freely mixed with braces but `{/}` pairs and `\begingroup/\endgroup` pairs should be properly matched. So `{ \begingroup \endgroup }` is allowed but `{ \begingroup } \endgroup` is not. Also it should be noticed that a brace group affects white-space when you're typesetting mathematics. This does not hold for `\begingroup/\endgroup`.

## 2.2 Diacritics

This section studies how typeset commonly occurring characters in combination with *diacritics*, which are also known as accents. Table 2.2 displays some commonly occurring diacritics and the commands which typeset them in E<sub>T</sub>X. The presentation is based on [Knuth, 1990, Chapter 9].

Using `\`{i}` to typeset *ï* may not work if you're not using a Type 1 font (T<sub>1</sub> font). However, typesetting *ï* with `\`{\i}` should always work. Here the command `\i` is used to typeset a dotless *i* (ı). There is also a command `\j` for a dotless *j*.

Table 2.3 displays some other commonly occurring special characters.

Table 2.3  
Other special characters.

Output	Command	Name
å	\aa	Scandinavian a-with-circle
Å	\AA	Scandinavian A-with-circle
ł	\l	Polish suppressed-l
Ł	\L	Polish suppressed-L
ø	\o	Scandinavian o-with-slash
Ø	\O	Scandinavian O-with-slash
¿	?‘	Open question mark
¡	!‘	Open exclamation mark

Table 2.4  
Foreign ligatures.

Output	Command	Name
œ	\oe	French ligature œ
Œ	\OE	French ligature Œ
æ	\ae	Latin and Scandinavian ligature æ
Æ	\AE	Latin and Scandinavian ligature Æ
ß	\ss	German ‘Eszett’ or sharp S

## 2.3 Ligatures

A *ligature* is a combination of two or several characters as a special glyph. Examples of English ligatures and the character combinations which they represent are fi (fi), ff (ff), ffi (ffi), fl (fl), and and ffl (ffl).  $\text{\LaTeX}$  recognises English ligatures and substitutes them for the character representing them.

Table 2.4 displays some foreign ligatures. The  $\text{\LaTeX}$  command which is required to typeset the symbol ß suggests that the es-zet is a ligature of ‘ss’. Indeed, this is how it is used. However, historically the symbol is a ligature of a long ‘s’ (ſ) and a ‘z’. Looking at the shape of the ligature, this sort of makes sense.

Sometimes it is better to suppress ligatures. The following is an example: it prevents  $\text{\LaTeX}$  from turning the ‘ff’ in ‘shelfful’ into a ligature, which makes the result much easier to parse.

He bought a shelf{}ful of books.  $\text{\LaTeX}$  Usage

If you use `xelatex` then the previous trick may not work but the following should.

He bought a shelf\hbox{}ful of books.  $\text{\LaTeX}$  Usage

## 2.4 Quotation Marks

This section explains how you typeset quotation marks. Figure 2.1 presents an example which is based on [Lamport, 1994, Page 13]. The word ‘Convention’ in this example is in single quotes and the word ‘this’ is in double quotes. The quotes at the start are backquotes (‘ and “). The

**Figure 2.1**  
Quotes.

---

‘Convention’ dictates that punctuation go inside quotes, like “this,” but some think it’s better to do “this”.

---



---

‘Convention’ dictates that punctuation go inside quotes, like “this,” but some think it’s better to do “this”.

---

**Figure 2.2**  
Nested quotations.

---

<code>“\, ‘Fi’ or ‘fum?’ \,” he asked. \\</code> <code>“‘Fi’ or ‘fum?’” he asked. \\</code> <code>“{ } ‘Fi’ or ‘fum?’ { }” he asked.</code>	<code>“ ‘Fi’ or ‘fum?’ ” he asked.</code> <code>“‘Fi’ or ‘fum?’” he asked.</code> <code>“‘Fi’ or ‘fum?’” he asked.</code>
---	---

---

quotes at the end are the usual quotes (‘ and ’). Notice that the quote between ‘it’ and ‘s’ is produced using a single quote in  $\text{\LaTeX}$ .

To get properly nested quotations you insert a *thin space* where quotes “meet”. You get a thin space with the command `\, ’`. Figure 2.2 provides a concrete example which is taken from [Lamport, 1994, Page 14]. Figure 2.2 provides another example. The first line of this example looks much better than the other two. Note that  $\text{\LaTeX}$  parses three consecutive quotes as a pair of quotes followed by one more quote. This is demonstrated by the second line of the output, which looks terrible. The last line of the input avoids the three consecutive quotes by adding an empty group which makes explicit where the double quotes and the single quote meet. Still the resulting output doesn’t look great.

**Intermezzo.** *As a general rule, British usage prefers the use of single quotes for ordinary use. This poses a problem with the single quote which is used for the possessive form: He said ‘It is John’s book.’ This is why it is also acceptable to use double quotes [Trask, 1997, Chapter 8].*

## 2.5 Dashes

There are three kinds of dashes: ‘-’, ‘--’, and ‘—’. In  $\text{\LaTeX}$  you get them with ‘-’, ‘--’, and ‘---’. The second symbol can also be typeset with the command `\textendash` and the last symbol with the command `\tex-temdash`. The symbol ‘—’, which is used in mathematical expressions such as ‘ $a - b$ ’, is not a dash. This symbol is discussed in Chapter 8. The following briefly explains how the dashes are used.

- This is the intra-word dash which is used to hyphenate compound modifiers such as one-to-one, light-green, and so on [Trask, 1997, Chapter 6]. In  $\text{\LaTeX}$  you typeset this symbol as follows: ‘-’.
- This is the *en-dash*, which roughly has a width of the letter N. It is mainly used in ranges: pages 12–15 (from 12 to 15). However, Allan [2001, page 45] notes that the en-dash is also used to link two names which are sharing something in common: a joint Anglo–French venture. The  $\text{\LaTeX}$  command `\textendash` and the sequence ‘--’ typeset the en-dash.



**Figure 2.3** The intra-word dash is used to hyphenate compound modifiers such as light-green, X-ray, or one-to-one. ...  
 Dashes The en-dash is used in ranges: pages~12--15.  
 The em-dash is used to separate strong interruptions from the rest of the sentence~---~like this%  
 ~\cite[Chapter~6]{Trask:1997}. ...

— This is the *em-dash*, which roughly has the same width as the letter M. It is used to separate strong interruptions from the rest of the sentence — like this [Trask, 1997, Chapter 6]. The  $\LaTeX$  command `\textemdash` and the sequence ‘`---`’ typeset the em-dash.

Figure 2.3 presents an example. A few years ago I noticed that sometimes ‘`---`’ doesn’t work in `xelatex` (even with ‘`Mapping = tex-text`’ enabled). However, `\textemdash` still works.

## 2.6 Periods

$\LaTeX$  usually treats a period (.) as an end-of-sentence indicator. By default,  $\LaTeX$  inserts a bit more space after the period at the end of a sentence than it does between words. Inserting the `\frenchspacing` turns this feature off. When a period is not the end of a sentence you need to help  $\LaTeX$  a bit by inserting the space command (`\`) after the period.

Mr.\~Happy has a compiler compiler named after him.  *$\LaTeX$  Usage*

However, when an uppercase letter is followed by a period, then  $\LaTeX$  assumes the period is being used for abbreviation. For example:

Donald E. Knuth developed the {\TeX} system.  *$\LaTeX$  Usage*

This convention causes a problem when an uppercase letter really is the end of a sentence. You insert the `\@` command before the period if this happens.

In Frank Herbert’s Dune saga,  
 the Mother School of the Bene Gesserit  
 is situated on the planet Wallach IX\@.

## 2.7 Emphasis

*Emphasis* is an typographic tool which typesets text in a different font from the rest of the text. The idea is that this makes the text stand out, thereby emphasising the text. Emphasis is especially useful to introduce new concepts, such as the first word in this paragraph.

In some documents, emphasis is implemented by typesetting text in a bold face font, by typesetting it in upper case font, or (worse) by

**Figure 2.4**  
Using footnotes.

---

```
Footnotes\footnote{A footnote is a note
of reference, explanation, or comment which is
usually placed below the text on a printed page.}
can be a nuisance. This is especially true if
there are many.\footnote{Like here.} The more
you see them, the more annoying they get.%
\footnote{Got it?}
```

Footnotes<sup>a</sup> can be a nuisance. This is especially true if there are many.<sup>b</sup>  
The more you see them, the more annoying they get.<sup>c</sup>

---

<sup>a</sup>A footnote is a note of reference, explanation, or comment which is usually placed below the text on a printed page.  
<sup>b</sup>Like here.  
<sup>c</sup>Got it?

---

underlining the text. In  $\text{\LaTeX}$ -produced documents, emphasised in-line text is usually slanted (italicised). Trask [1997, Page 82] states that this is the preferred style for emphasis.

In  $\text{\LaTeX}$  emphasis is implemented with the command `\emph`. emphasised in-line text The best way to regard the command `\emph` is as an emphasis switch, which switches from upright to slanted and back. The semantics of the declaration depend on how many times it is active.

- If, at the current position, the number of active applications is even — this includes zero — then the current style is upright, and the next application switches to a slanted style.
- Otherwise, the current style is slanted, and the next application switches to upright.

```
\emph{An
\emph{example
\emph{with}
    nested
\emph{emphasis}}}.
An example with nested emphasis.
```

## 2.8 Footnotes and Marginal Notes

It is generally accepted that footnotes and marginal notes should be used sparingly because they are disruptive. However, proper use of marginal notes in documents with wide margins can be very effective.

Not surprisingly,  $\text{\LaTeX}$  provides a command for footnotes and a command for marginal notes. Figure 2.4 demonstrates how to specify footnotes in  $\text{\LaTeX}$ . A *marginal note* or *marginal paragraph* is like a footnote, but then in the margin as on this page. The command `\marginpar{<text>}` puts `<text>` in the margin as a marginal note. By passing an optional argument to the command you can put different text on odd (recto/front/right-hand) pages and on even (verso/back/left-hand) pages. The optional argument is used for even pages and the required argument

In texts with narrow margins it is better to avoid marginal notes.

**Figure 2.5**  
The quote environment.

---

```

Blah blah blah blah blah blah blah blah blah.
\begin{quote}
  Next to the originator of a good sentence
    is the first quoter of it. \\
  \emph{Ralph Waldo Emerson}
\end{quote}
Blah blah blah blah blah blah blah blah blah.

```

---

Blah blah blah blah blah blah blah blah blah.

Next to the originator of a good sentence is the first quoter  
of it.  
*Ralph Waldo Emerson*

Blah blah blah blah blah blah blah blah blah.

---

is used for odd pages. If you're using both the optional and required argument then it is easy to remember which is which: the optional argument is to the left of the required argument so it's for the left-hand page; the required argument is for the right-hand page. Note that marginal notes may look better with ragged text.

## 2.9 Displayed Quotations and Verses

The quote and quotation environments are for typesetting displayed quotations. The former is for short quotations; the latter is for longer quotations. Figure 2.5 shows how you use the quote environment. The command ‘\\’, which is used in Figure 2.5, forces a line break.

The verse environment typesets poetry and verse. Figure 2.6 shows how you use the environment. In this example, the command \qqquad inserts two *quads*. Here a *quad* is an amount of space which is equivalent to the size of the current font. So if you're using a 12 pt font then a quad results in a 12 pt space. This space is about the same as the width of the letter ‘M’. As with the quote environment, the verse environment also required that line breaks are specified explicitly with the command ‘\\’.

## 2.10 Line Breaks

In the previous section the \\ inserted a line break in displayed quotations and verses. The command also works inside paragraphs. The command takes an optional argument which determines the extra vertical space of the line break: \\[⟨extra vertical space⟩]. A line break at the end of a page may trigger page break. If page breaks aren't desirable you should use the command ‘\\\*’, which works similar to \\, except that it inhibits page breaks.

**Figure 2.6**  
The verse environment.

---

The following anti-limerick is attributed to W. S. Gilbert.

```
\begin{verse}
  There was an old man of St. Bees,      \\
  Who was stung in the arm by a wasp;    \\
    \quad When they asked, "Does it hurt?" \\
    \quad He replied, "No, it does n't,    \\
  But I thought all the while 't was a Hornet."
\end{verse}
```

---

The following anti-limerick is attributed to W. S. Gilbert.

There was an old man of St. Bees,  
Who was stung in the arm by a wasp;  
    When they asked, "Does it hurt?"  
    He replied, "No, it does n't,  
But I thought all the while 't was a hornet."

---

**Table 2.5**  
Size-affecting declarations  
and environments.

Declaration	Environment	Example
<code>\tiny</code>	<code>tiny</code>	Example
<code>\scriptsize</code>	<code>scriptsize</code>	Example
<code>\footnotesize</code>	<code>footnotesize</code>	Example
<code>\small</code>	<code>small</code>	Example
<code>\normalsize</code>	<code>normalsize</code>	Example
<code>\large</code>	<code>large</code>	Example
<code>\Large</code>	<code>Large</code>	Example
<code>\LARGE</code>	<code>LARGE</code>	Example
<code>\huge</code>	<code>huge</code>	Example
<code>\Huge</code>	<code>Huge</code>	Example

## 2.11 Controlling the Size

With the proper class and packages there usually is no need to change the type size of your text. However, sometimes it has its merits, e.g., when you're designing your own titlepage or environment. Table 2.5 lists the declarations and environment which let you to change the type size. The preferred "size" for long-ish algorithms and program listings is `\scriptsize`. If you're using a package to typeset your listings, the package usually chooses the right size for you. If not, it probably lets you specify the type size. Figure 2.7 shows how you change the size of text.

**Figure 2.7**  
Controlling the size.

```
{\tiny Mumble.      \\
\begin{normalsize}
    What?
\end{normalsize}    \\
\begin{Huge}
    I said ‘Mumble’.
\end{Huge} }
```

Mumble.  
What?  
**I said ‘Mumble’.**

**Table 2.6**

Type style affecting declarations and commands. The main font which is used to typeset the text in the last column is *Computer Modern*, which is what most L<sup>A</sup>T<sub>E</sub>X users are used to. The four lines at the top of the table usually correspond to the default style. The first nine type faces have *proportional* widths, which means that dif-

Declaration	Command	Example
<code>\mdseries</code>	<code>\textmd</code>	Medium Series
<code>\normalfont</code>	<code>\textnormal</code>	Normal Style
<code>\rmfamily</code>	<code>\textrm</code>	Roman family
<code>\upshape</code>	<code>\textup</code>	Upright Shape
<code>\itshape</code>	<code>\textit</code>	<i>Italic Shape</i>
<code>\slshape</code>	<code>\textsl</code>	<i>Slanted Shape</i>
<code>\bfseries</code>	<code>\textbf</code>	<b>Boldface Series</b>
<code>\scshape</code>	<code>\textsc</code>	SMALL CAPS SHAPE
<code>\sffamily</code>	<code>\textsf</code>	Sans Serif Family
<code>\ttfamily</code>	<code>\texttt</code>	Typewriter Family

ferent glyphs may have different widths. Typical examples of glyphs with different widths are the upper case ‘M’ and the lower case ‘i’. In the typeface on the last line, each character has the same width. These fonts/typefaces are called *non proportional* or *monospaced*. They are very useful for typesetting program listings.

## 2.12 Controlling the Type Style

Whereas changing the type size is hardly ever needed, changing the type style is required more often. There are ten L<sup>A</sup>T<sub>E</sub>X declarations which affect the type style. For each declaration there is a corresponding command which takes an argument and applies the type style of the declaration to the argument. The declarations and commands are listed in Table 2.6.

**Intermezzo.** *If you do need to change the type style of your text then it is probably for a specific purpose. For example, changing the type style of an identifier in an algorithm, changing the type style of a constant in an algorithm, and so on. Rather than hard-coding the style when you typeset your text, it is better to (1) define a user-defined command that typesets your text in the required style, and (2) use the command to typeset your text. The advantage of doing things this way is that it improves maintainability. For example, if you want to change the type style of all identifiers in your text then you only need to make changes in the definition of the command that typesets identifiers. Defining your own commands is discussed in Chapter 10.*

## 2.13 Phantom Text

This section is devoted to commands which don’t typeset anything but which do affect the horizontal and vertical spacing.

<b>Figure 2.8</b> The <code>\phantom</code> command.	Fill in the missing word.\\ Fill in the missing <code>\phantom{word}</code> .	Fill in the missing word. Fill in the missing .
---	---	--

`\phantom{⟨stuff⟩}`

This command “typesets” its argument using invisible ink. The dimensions of the box are the same as the dimensions required for typesetting `⟨stuff⟩`. “The font” which is used to typeset the invisible text is the same as the current font. □

Figure 2.8 demonstrates how you use the command.

The `\hphantom` and `\vphantom` commands are horizontal and vertical versions of the `\phantom` command. The following explains how they work.

`\hphantom{⟨stuff⟩}`

This is the horizontal version of the `\phantom` command. The command creates a box with zero height and the same width as its argument, `⟨stuff⟩`. □

`\vphantom{⟨stuff⟩}`

This is the vertical version of the `\phantom` command. The command creates a box with zero width and the same height as its argument, `⟨stuff⟩`. It is especially useful for getting the right size for delimiters such as parentheses in mathematical formulae that span multiple lines. This is explained in more detail in Section 8.8.1. □

## 2.14 Alignment

This section studies three commands and two environments which change the text alignment. The first command centres text. The second and third command align text to the left and to the right. The first of the environments is the `tabular` environment, which typesets row-based content with vertical alignment. The last environment is the `tabbing` environment. This environment lets you define vertical alignment (`tab`) positions and lets you position horizontal text relative to these alignment positions.

### 2.14.1 Centred Text

Centering text is done with the `center` environment. The following example is inspired by Iggy Pop.

<code>\begin{center}</code>	
Blah.\\	Blah.
Blah blah blah.	Blah blah blah.
Blah blah blah blah blah	Blah blah blah blah blah blah
blah blah blah blah blah	blah blah blah blah blah blah
blah blah blah blah blah.	blah blah.
<code>\end{center}</code>	

## 2.14.2 Flushed/Ragged Text

The `flushleft` environment and the `\raggedright` declaration typeset text which is aligned to the left. Likewise, the `flushright` environment and `\raggedleft` declaration typeset text which is aligned to the right. The following shows the effect of the `flushleft` environment.

```
\begin{flushleft}
  Blah.\\
  Blah blah blah.

  Blah blah blah blah blah
  blah blah blah blah blah
  blah blah blah blah blah.
\end{flushleft}
```

Blah.	Blah.
Blah blah blah.	Blah blah blah.
Blah blah blah blah blah	Blah blah blah blah blah blah
blah blah blah blah blah	blah blah blah blah blah blah
blah blah blah blah blah.	blah blah blah blah blah.

## 2.14.3 Basic tabular Constructs

The `tabular` environment typesets text with rows and columns with vertical alignment. The environment also has siblings called `tabular*` and `array`. The `tabular*` environment works similar to `tabular` but it takes an additional argument which determines the width of the resulting construct. This environment is explained in Section 2.14.5. The `array` environment can only be used in math mode. It is explained in Chapter 8. The `tabular` and `tabular*` environments can be used in text and math mode.

The remainder of this section is an introduction to the `tabular` environment. This introduction should more than likely suffice for day-to-day usage. A more detailed presentation is provided in Section 2.14.5.

In its simplest form the `tabular` environment is used as follows.

```
\begin{tabular}[<global alignment>]{<column alignment>}
  <text> & <text> & ... & <text> \\
  ...
  <text> & <text> & ... & <text> \\
  <text> & <text> & ... & <text>
\end{tabular}
```

The body of the environment contains a sequence of rows which are delimited by linebreaks (`\\`). Each row is a sequence of alignments tab-delimited `<text>`. The  $i$ -th `<text>` in a row corresponds to the  $i$ -th column. The following explains the arguments of the environment:

`<global alignment>`

This optional argument determines the vertical alignment of the environment. Allowed values are 't' (align on the top row), 'c' (align on the centre), or 'b' (align on the bottom row). This default value of this argument is 'c'. □

`<column alignment>`

This argument determines the alignment of the columns and additional

decorations. For day-to-day usage, the following options are relevant.

- 1 This option corresponds to a column. The column is left-aligned.
- r This also corresponds to a column. The column is right-aligned.
- c This also corresponds to a column. The column is centred.
- p{<width>} This option corresponds to a top-aligned <width>-wide column which is typeset as a paragraph in the “usual” way. Some commands such as \ are not allowed at the top level. See [Lamport, 1994, Appendix C.10.2] for further information.
- | This options does not correspond to an actual column but results in additional decoration. It results in a vertical line which is drawn at the “current” position. For example, if <column alignment> is ‘l|cr’ then there will be a vertical line separating the first two columns. Using this option is discouraged because vertical lines are usually distracting. □

The tabular environment also defines commands which may be used inside the environment.

\hline

This command inserts a horizontal rule. The command may only be used at the *start* of a row. □

\cline{<number>\_1}{<number>\_2}

This draws a horizontal line from the start of column <number>\_1 to the end of column <number>\_2. □

\vline

This command results in a vertical line. The command may only be used if the column is aligned to the left, to the right, or to the centre. In addition, it may be used in a so-called *@-expression*, which is explained in the next section. □

Figure 2.9 presents a simple example of the tabular environment. The example exhibits all possible alignments as well as the paragraph “alignment” feature. Note that line breaks are inserted inside the ‘p’ column. This is never done if a column is aligned with ‘l’, ‘r’, and ‘c’.

**Intermezzo.** *The column alignment option ‘l’ and the commands \hline, \cline, and \vline are irresistible to new users. I suspect this is because most tabular examples involve the option and these commands. It is understandable that new users want to repeat this, especially if they’re not aware that using the option and the commands in moderation is better because the resulting grid lines are dazzling and distracting. Chapter 6 provides some guidelines on how to design good tables.*

Regular  $m \times n$  tables with the same alignment in the same column are rare. The following command lets you join columns within a row and override the default alignment.



**Figure 2.9**  
Using the tabular environment.

```
\begin{tabular}{l|crp{3.1cm}}
\hline
1 & 2 & 3
& Box me in, but witho%
ut those awkward line
breaks, please.
\\ \hline
11 & 12 & 13
& Excellent.
\\ 111 & 112 & 113
& Thank you!
\\ \hline
\end{tabular}
```

1	2	3	Box me in, but without those awkward line breaks, please.
11	12	13	Excellent.
111	112	113	Thank you!

`\multicolumn{⟨number⟩}{⟨column alignment⟩}{⟨text⟩}`

This inserts `⟨text⟩` into a *single* column which is formed by combining the next `⟨number⟩` columns in the current row. The alignment of the resulting column is determined by `⟨column alignment⟩`. This command is especially useful to override the default alignment in column headings of a table. An example involving the command is presented in the next section. □

## 2.14.4 The booktabs Package

The `booktabs` package adds some extra functionality to the `tabular` environment. The package discourages vertical grid lines. Using the `booktabs` package results in better looking tables.

- The package provides different commands for different rules.
- The package provides different rules which may have different widths.
- The package provides commands for temporarily/permanently changing width.
- The package has a command which adds extra line space.
- The package is compatible-ish with the `colortbl` package which is used to specify coloured tables.

The `booktabs` package provides the following commands for rules. The first four commands take an optional argument specifying the width of the rule.

`\toprule[⟨width⟩]`

This typesets the full horizontal rule at the top the table. □

`\bottomrule[⟨width⟩]`

This typesets the full horizontal rule at the bottom of the table. □

`\midrule[⟨width⟩]`

This typesets the remaining full horizontal rules in the table. □

**Figure 2.10**  
Using booktabs rules.

```
\begin{tabular}[c]{lr@{ }lr@{.}lp{47mm}}
\toprule \multicolumn{1}{l}{\textbf{Destination}}
& \multicolumn{1}{l}{\textbf{Duration}}
& \multicolumn{2}{l}{\textbf{Price}}
& \multicolumn{1}{l}{\textbf{Description}}
\\ \midrule
Cork City
& 7 & Days & \euro 300 & 00
& Visit Langer Laand. Price includes visits
to Rory Gallagher Place and de Maarkit.
\\ Ballinspittle
& 8 & Days & \euro 400 & 00
& Price includes visit to statue of
the Blessed Virgin Mary. See it move!
\\ \bottomrule
\end{tabular}
```

**Table 2.7**

This table demonstrates the rules provided by the booktabs package. The input for this table is listed in Figure 2.10. Clearly, booktabs rules.

Destination	Duration	Price	Description
Cork City	7 Days	€300.00	Visit Langer Laand. Price includes visits to Rory Gallagher Place and de Maarkit.
Ballinspittle	8 Days	€400.00	Price includes visit to statue of the Blessed Virgin Mary. See it move!

`\cmidrule[⟨width⟩]{⟨number⟩1–⟨number⟩2}`

This typesets partial horizontal rules in the middle of the table. The rule ranges from the start of column  $\langle \text{number} \rangle_1$  the end of column  $\langle \text{number} \rangle_2$ .

□

`\addlinespace[⟨width⟩]`

This command is usually used immediately after a line break and (guess) it gives you more vertical line space. □

Figure 2.10 demonstrates how to use the booktabs-provided rule commands. The resulting output is presented in Table 2.7. Notice that the inter-linespacing is much better than the output in Figure 2.9. Also notice the different widths of the rules.

## 2.14.5 Advanced tabular Constructs

Using basic tabular constructs usually suffices for day-to-day typesetting. This section explains the techniques which give you the power to typeset more advanced tabular constructs.

The following starts by presenting two addition column options. This is followed by some style parameters which control the default size and spacing of the tabular, tabular\*, and array environments. The column options are as follows.

**Figure 2.11**

Controlling column widths with an @-expression. The output is stretched out for clarity.

```
\begin{tabular*}{3cm}{@{}lcr@{}}
\toprule M & M & M \\\bottomrule
\end{tabular*}

\begin{tabular*}{3cm}
{@{\extracolsep{\fill}}%
lcr%
@{\hspace{0pt}}}%
\toprule M & M & M \\\bottomrule
\end{tabular*}

\begin{tabular*}{3cm}
{@{\hspace{\tabcolsep}}%
@{\extracolsep{\fill}}%
lcr%
@{\hspace{\tabcolsep}}}%
\toprule M & M & M \\\bottomrule
\end{tabular*}
```

M	M	M	M	M	M	M	M	M
---	---	---	---	---	---	---	---	---

`*{<number>}{<column options>}`

This inserts `<number>` copies of `<column options>`. For example, `*{2}{lr}` is equivalent to `lr`. □

`@{<text>}`

This is called an @-expression. It inserts `<text>` at the current position. This may be useful if you want to add certain text or symbols at the given position. For example `@{.}` inserts a period at the current position.

TeX normally inserts some horizontal space before the first column and after the last column. It inserts twice that amount of space between adjacent columns. However, this space is suppressed if an @-expression precedes or follows a column option. For example, if `<column alignment>` is equal to `@{}ll@{}l@{}l` then this suppresses the horizontal space before the first column, after the last column, and between the second and last column. The length `\tabcolsep` controls the extra horizontal space which is inserted. The value of the command is half the width which is inserted between columns.

A horizontal spacing command in an @-expression controls the separation of two adjacent columns. For example, `@{\hspace{<width>}}` inserts a horizontal `<width>`-wide space.

Finally, @-expressions may also adjust the default column separation. Using `\extracolsep{<width>}` adds additional horizontal `<width>`-wide space between subsequent columns. *However*, additional width is never inserted before the first column. Using `\extracolsep{\fill}` inserts the maximum possible amount of horizontal space. This is useful if you want to extend the width to the maximum possible width.

Figure 2.11 demonstrates an application of an @-expression the the maximal stretching of columns in a `tabular*` environment. □

The following commands control the default appearance of `tabular`, `tabular*`, and `array` environments.

<code>\arraycolsep</code>	The value of this length command is equal to half the default horizontal distance between adjacent columns in the <code>array</code> environment. This amount of space is also equal to the default horizontal space which is inserted before the first column and after the last column. <input type="checkbox"/>
<code>\tabcolsep</code>	The value of this length command is equal to half the default horizontal distance between adjacent columns in the <code>tabular</code> and <code>tabular*</code> environments. Again, this is equal to the default horizontal space which is inserted before the first column and after the last column. <input type="checkbox"/>
<code>\arrayrulewidth</code>	The value of this length command is the width of the lines resulting from a ‘ ’ in the <code>&lt;column options&gt;</code> argument and the lines resulting from the commands <code>\cline</code> , <code>\hline</code> , and <code>\vline</code> . <input type="checkbox"/>
<code>\doublerulesep</code>	The value of this length command is the distance between two adjacent lines resulting from a ‘  ’ in the <code>&lt;column options&gt;</code> argument or two adjacent lines resulting from the <code>\hline</code> command. <input type="checkbox"/>
<code>\arraystretch</code>	This command determines the distance between successive rows. It defaults to 1 and “multiplying” it by $x$ results in rows which are $x$ times further apart. So, by redefining this command to 0.50 you halve the distance between successive rows. Redefining commands is explained in Chapter 10. <input type="checkbox"/>

### 2.14.6 The tabbing Environment

The tabbing environment is useful for vertical alignment relative to user-definable alignment positions. The remainder of this section describes some basic usage of the environment. The reader is referred to [Lamport, 1994, pages 201–203] for more detailed information.

The tabbing environment can only be used in *paragraph mode* (the “usual mode”). It produces lines of text with alignment in columns based upon *tab positions*.

<code>\=</code>	Defines the next tab (vertical alignment) position. <input type="checkbox"/>
<code>\\</code>	Inserts line break and resets next tab position to <code>left_margin_tab</code> . <input type="checkbox"/>
<code>\kill</code>	Throws away the current line but remembers the tab positions defined with <code>\=</code> . <input type="checkbox"/>
<code>\+</code>	Increments <code>left_margin_tab</code> . <input type="checkbox"/>
<code>\-</code>	Decrements <code>left_margin_tab</code> . <input type="checkbox"/>

**Figure 2.12**

The tabbing environment.

<code>\begin{tabbing}</code>	
From <code>\=here</code> to <code>\=there</code> <code>\\</code>	From here to there
<code>\&gt;and</code> <code>\&gt;then</code> <code>\\</code>	and then
<code>\&gt;\&gt;all</code> <code>\\</code>	all
<code>\&gt;the</code> <code>\&gt;way</code> <code>\\</code>	the way
back <code>\&gt;to</code> <code>\&gt;here.</code>	back to here.
<code>\end{tabbing}</code>	

**Figure 2.13**

Advanced use of tabbing environment.

<code>\begin{tt}\begin{tabbing}</code>	
AAA <code>\=AAA\=AAA\=AAA</code> <code>\kill</code>	
<code>FUNC euc( INT a,</code>	<code>FUNC euc( INT a, INT b ) : INT</code>
<code>INT b ) : INT</code> <code>\\</code>	<code>BEGIN</code>
<code>BEGIN \+ \\</code>	<code>WHILE (b != 0) DO</code>
<code>WHILE (b != 0) DO</code> <code>\\</code>	<code>BEGIN</code>
<code>BEGIN \+ \\</code>	<code>INT rem = a MOD b;</code>
<code>INT rem = a MOD b;</code> <code>\\</code>	<code>a = b;</code>
<code>a = b;</code> <code>\\</code>	<code>b = rem;</code>
<code>b = rem;</code> <code>\- \\</code>	<code>END</code>
<code>END</code> <code>\\</code>	<code>RETURN a;</code>
<code>RETURN a;</code> <code>\- \\</code>	<code>END;</code>
<code>END;</code>	
<code>\end{tabbing}\end{tt}</code>	

`\>`

Move to the next tab stop.

□

Figures 2.12 and 2.13 present two examples of the tabbing environment. The examples do not demonstrate the full functionality of the environment.

## 2.15 Language Related Issues

As suggested by its title, this section is concerned with language related issues. The remaining three sections deal with hyphenation, foreign languages, and spelling.

### 2.15.1 Hyphenation

$\text{\LaTeX}$ 's ( $\text{\TeX}$ 's really) automatic hyphenation is second to none. However, sometimes even  $\text{\TeX}$  gets it wrong. There are two ways to overcome such problems.

- The command `\-` in a word tells  $\text{\LaTeX}$  that it may hyphenate the word at that position.

Er`\-go``\-no``\-mic` has  
three hyphenation positions.

$\text{\LaTeX}$  Usage

- Specifying the same hyphenation patterns is messy and prone to errors. Using the `\hyphenation` command is a much cleaner

**Figure 2.14** Using the babel package.

---

```
\usepackage[dutch,british]{babel}
:

\selectlanguage{dutch}
% Dutch text here.
Nederlandse tekst hier.

\selectlanguage{british}
% Engelse tekst hier.
English text here.
```

---

solution. This command takes one argument, which should be a comma-separated list of words. For each word you can put a hyphen at the (only) possible/desired/allowed hyphenation positions. You may use the command several times. The following is an example.

```
\hyphenation{fortran,er-go-no-mic}
```

*L<sup>A</sup>T<sub>E</sub>X* Usage

### 2.15.2 Foreign Languages

The babel package supports multi-lingual documents. The package supports proper hyphenation, switches between different languages in one single document, definition of foreign languages, commands which recognise the “current” language, and so on. Figure 2.14 provides a minimal example. Rik Kabel kindly informed me that xelatex users use the polyglossia package instead of babel. One of the advantages of the polyglossia package is that it automatically loads the bidi package when bi-directional scripts are used.

### 2.15.3 Spell-Checking

*L<sup>A</sup>T<sub>E</sub>X* does not support automatic spell-checking. Note that spell-checking is not exactly non-trivial because text may be generated. In addition text may come from external files: spell-check your bibliography files too. The vim program has a spell-checker plugin which is called SpellChecker. The ispell program supports *L<sup>A</sup>T<sub>E</sub>X*. The ‘-t’ flag tells the command that the input is *L<sup>A</sup>T<sub>E</sub>X*.

```
$ ispell -l -t -S input.tex | sort -u
```

Unix Session

# CHAPTER 3

## Lists

THIS CHAPTER is about *lists*. Here, a *list* is a sequence of labelled *items*.  $\text{\LaTeX}$  has three built in environments supporting *unordered* lists, *ordered* lists, and *description* lists.

$\text{\LaTeX}$  marks each item in the output list by giving it a *label* which precedes the item. The items are typeset in paragraphs with hanging indentation, which means the paragraphs are indented a bit further than the surrounding text.

In *unordered* lists, the order of the items is irrelevant-ish and all labels are the same. Usually the labels are bullet points, asterisks, dashes, .... Most people refer to such lists as bullet points.

In an *ordered* list the order of the items *does* matter. Each label indicates the order of its item in the list. You could say that the items are numbered by the labels. Usually, the labels are arabic numbers (1, 2, 3, ...), lower case roman numerals (i, ii, iii, ...), lower case letters (a, b, ..., z), and so on.

In *description* lists, the order of the items may or may not matter. In such lists, each label is a (short) description of its item.

The remainder of this chapter is as follows. Section 3.1 explains unordered lists. Typesetting ordered (also enumerated) lists is studied in Section 3.2. Section 3.3 describes the `enumerate` package, which provides a high-level interface to control the labels which are used for the lists. Section 3.4 shows how you typeset description lists. Section 3.5, which is the icing on the cake, describes how to create your own lists.

### 3.1 Unordered Lists

The `itemize` environment is for creating *unordered lists*. In the body of the environment you start each item in the list using the `\item` command. Each `itemize` environment should have at least one `\item`. Nested itemised lists are possible, but the nesting level is limited. Figure 3.1 shows how you use the `itemize` environment.

Each item in the list is preceded by its label. Usually, the shape of the top-level label is a bullet point but the shape may depend on your document class and your packages.

The command `\labelitemi` determines the shape of the label of

**Figure 3.1**

The `itemize` environment. Notice that the labels of the nested list are different from the labels of the top-level list.

---

```

\begin{itemize}
\item First item.
\item Second item.
    Text works as usual here.
\item Third item is a list.
    Different labels here.
    \begin{itemize}
    \item First nested item.
    \item Second item.
    \end{itemize}
\end{itemize}

```

---

- First item.
- Second item. Text works as usual here.
- Third item is a list. Different labels here.
  - First nested item.
  - Second item.

---

**Figure 3.2**

Changing the item label. The default label for top-level itemised lists is the bullet. The labels of the itemised top-level list which is defined in the group is a plus sign. This different label is the result of redefining the `\labelitemi` command inside the group. By redefining the command inside the group the new

---

```

{% Start a group.
\renewcommand{\labelitemi}{+}
\begin{itemize}
\item Label is plus.
\end{itemize}
}% End of group.
\begin{itemize}
\item Default label.
\end{itemize}

```

---

- + Label is plus.
- Default label.

---

definition of the command is kept local to the group. The original definition of the command is restored upon leaving the group. This explains why the labels of the second itemised list are bullets.

the top-level items. Likewise, `\labelitemii` is for labels of subitems, `\labelitemiii` for the labels of subsubitems, and `\labelitemiv` for the labels of subsubsubitems. By redefining these commands, you may change the appearance of the labels. You may redefine an existing command with the `\renewcommand` command. The following sets the shape of the labels at the top level to a plus sign and the labels at level four to a minus sign. The command `\renewcommand` is discussed in more detail in Chapter 10.

```

\renewcommand{\labelitemi}{+}
\renewcommand{\labelitemiv}{-}

```

*LaTeX Usage*

If you only want to change the appearance of a given label for a single list then the easiest way is to redefine the `\labelitemi` command a *group* and put the list in that group. This keeps the new definition of `\labelitemi` local to the group.<sup>1</sup> Remember that you may create a group using braces. Figure 3.2 shows how to change the appearance of the label at Level 1 and keep the change local to the group.

<sup>1</sup> See Section 2.1.2 for further information about the merits of groups.



**Figure 3.3**

The `enumerate` environment. Notice that the labels of the top-level list are numeric, whereas the labels of the nested list are parenthesised lower-case letters.

---

```

\begin{enumerate}
\item First item.
\item Second item.
\item Third item is a list.
    \begin{enumerate}
    \item First nested item.
    \item Second item.
    \end{enumerate}
\end{enumerate}

```

---

1. First item.
2. Second item.
3. Third item is a list.
  - (a) First nested item.
  - (b) Second item.

---

## 3.2 Ordered Lists

The `enumerate` environment is for creating ordered lists. It works just as the `itemize` environment but this time the labels are numbers, letters, or roman numerals, and the like. Figure 3.3 demonstrates the environment.

As with the appearance of the labels in the `itemize` environment you may also change the appearance of the labels in the `enumerate` environment. This time the appearance depends on the four labelling commands `\labelenumi`, `\labelenumii`, `\labelenumiii`, and `\labelenumiv`. Each of these commands depends on a *counter* which counts the items at its level. Counters are explained in further detail in Chapter 12. The top level items are counted using the counter `enumi`, the second level items with `enumii`, the third level items with `enumiii`, and the fourth level items with `enumiv`. These counters are managed by the labelling commands. When a labelling command typesets its label, it uses the corresponding counter and typesets the label in a certain style. The style is hard-coded in the command. Typical styles are (1) arabic numbers, (2) lowercase or uppercase roman numerals, and (3) lowercase or uppercase letters. Implementing the typesetting of the label in these styles is done with the commands `\arabic`, `\roman`, `\Roman`, `\alph`, and `\Alph`, which typeset their counter using arabic numbers, lower case roman numerals, upper case roman numerals, lower case letters, and upper case letters. The following demonstrates how you get lower case roman numerals for the labels at the top level and numbers for the labels at the third level.

```

\renewcommand{\labelenumi}{\roman{enumi}}
\renewcommand{\labelenumiii}{\arabic{enumiii}}

```

*ETEX Usage*

## 3.3 The `enumerate` Package

Having to redefine the labelling commands is tedious and prone to error. The `enumerate` package provides a high-level interface to  $\text{\TeX}$ 's default mechanism for selecting the labels of enumerated lists. Basically, the package redefines the `enumerate` environment. The resulting environment has an optional argument which determines the style of the labels

**Figure 3.4** Surrounding text here.

Using the `enumerate` package. The enumerated list in this example is created with the environment `enumerate`, which is redefined by the `enumerate` package. The optional argument of the environment defines labels of the form ‘Item-⟨upper case

letter⟩’. Putting the text ‘Item’ inside the braces tells the `enumerate` environment that it should not interpret the letters in the text as labels. Notice that the label is typeset relative to the start of the hanging paragraphs. For short labels this is not a problem. However, with long labels, such as in this example, this results in the label protruding into the margin of the surrounding text.

---

<code>\usepackage{enumerate}</code>	Surrounding text here.
<code>\begin{enumerate}</code>	
<code>[\textbf{{Item}-A}]</code>	<b>Item-A</b> The first of two hanging paragraphs.
<code>\item The first of two hanging paragraphs.</code>	
<code>\item The second of hanging paragraphs.</code>	<b>Item-B</b> The second of two hanging paragraphs.
<code>\end{enumerate}</code>	

---

**Figure 3.5** Kurasawa films include:

Using the `description` environment. The environment works almost the same as the `itemize` and `enumerate` environments. The only difference is that this time you provide the labels for the list as optional arguments of the `\item` command.

---

<code>\begin{description}</code>	Kurasawa films include:
<code>\item[Kagemusha]</code>	<b>Kagemusha</b> When a powerful warlord in medieval Japan dies, a poor thief is recruited to impersonate him. ...
<code>\item[Yojimbo]</code>	<b>Yojimbo</b> A crafty ronin comes to a town divided by two criminal gangs. ...
<code>\item[Sanshiro Sugata]</code>	<b>Sanshiro Sugata</b> A young man, struggles to learn the nuance and meaning of judo. ...
<code>\end{description}</code>	...

---

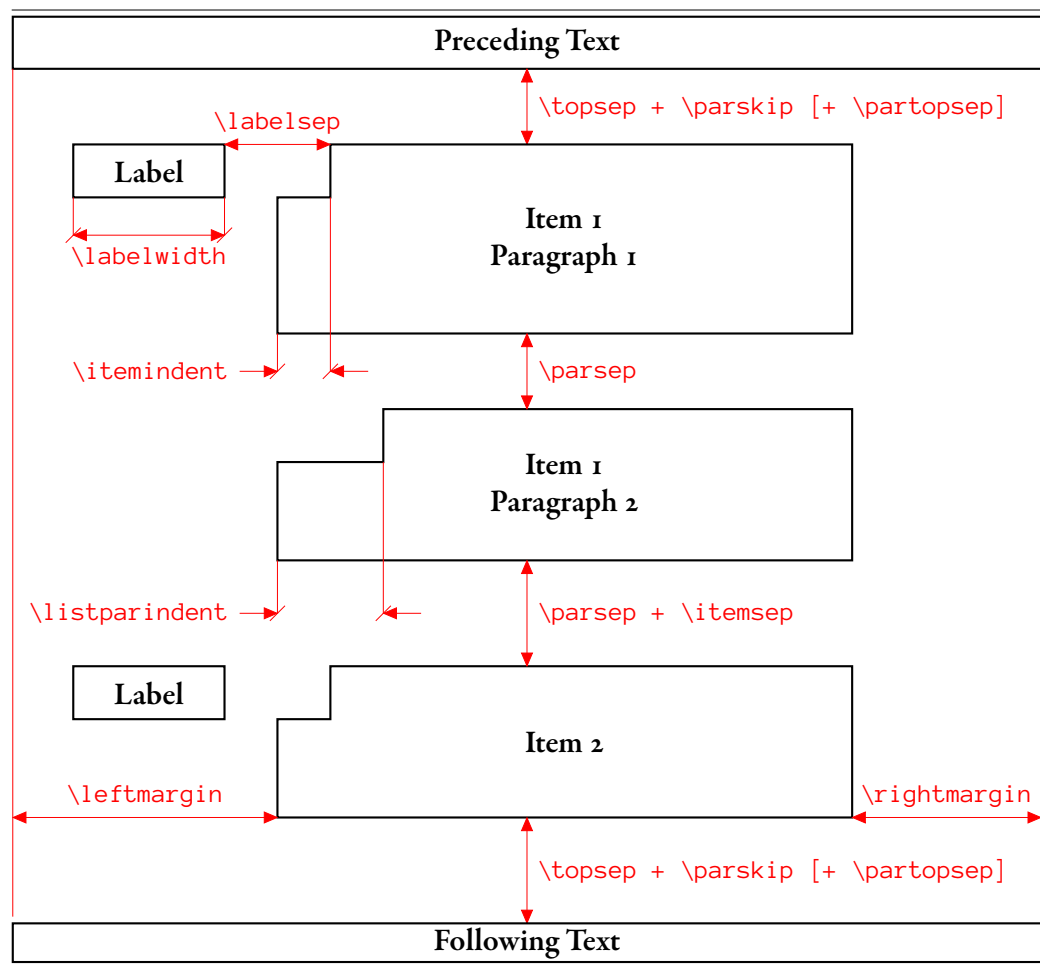
of the lists. For example, using the option ‘A’ results in labels which are typeset using the command ‘`\Alph`’. Likewise the options ‘a’, ‘I’, ‘i’, and ‘1’ result in labels which are typeset using the commands ‘`\alph`’, ‘`\Roman`’, ‘`\roman`’, and ‘`\arabic`’.

However, the package is more flexible and also allows you to specify different kinds of labels. Figure 3.4 provides an example. The interested reader is referred to the package documentation [D. Carlisle, 1999a] for further details.

### 3.4 Description Lists

The `description` environment is for creating labelled lists. The labels are passed as optional arguments to the `\item` command. Figure 3.5 provides an example of how to use the `description` environment.

**Figure 3.6**  
Lengths which affect the formatting of a  $\text{\LaTeX}$  list environment.



### 3.5 Making your Own Lists

$\text{\LaTeX}$ 's list environment lets you define your own lists.

```
\begin{list}{\langle label commands \rangle}{\langle formatting commands \rangle} \langle item list \rangle \end{list}
```

Here  $\langle label formatting commands \rangle$  typesets the labels. For ordered lists you may need to define a dedicated counter that keeps track of the numbers of the labels. How to do this is explained further on. The  $\langle list formatting commands \rangle$  format the resulting list. The formatting depends on *length* commands. These commands determine lengths and widths which are used to construct the resulting list. For example, the distance between adjacent items, the distance between a label and its item, and so on. Figure 3.6 depicts the relevant length commands and how they determine the formatting of the list. The picture is based on [Lamport, 1994, Figure 6.3]. The horizontal length commands are *rigid* lengths. As the name suggests the resulting dimensions are fixed. The vertical length commands are *rubber* lengths. The resulting dimensions may shrink or stretch depending on the lack or excess of vertical space on the page.  $\square$

Figure 3.7 presents an example of a user-defined list. The command `\newcounter{ListCounter}` defines a new counter called `ListCounter`. The spell `List-\alph{ListCounter}` typesets the label of each item as

**Figure 3.7**  
A user-defined list.

---

```

\newcounter{ListCounter}

\begin{list}
  {List-\alph{ListCounter}}
  {\usecounter{ListCounter}
   \setlength{\rightmargin}{0cm}
   \setlength{\leftmargin}{2cm}}
\item Hello.
\item World.
\end{list}

```

---

**Figure 3.8**  
A user-defined environment for lists. In this example, the `\newenvironment` command takes three arguments. The first is the name of the environment, the second argument determines what to do upon entering the environment, and the third what to do upon leaving the environment. The second argument opens a simple `list` environment and the third closes the `list` environment.

---

```

\newcounter{ListCounter}
...
% Define new environment:
\newenvironment
  {alphList}
  {\begin{list}
   {List-\alph{ListCounter}}
   {\usecounter{ListCounter}
    \setlength{\rightmargin}{0cm}
    \setlength{\leftmargin}{2cm}}}
  {\end{list}}
...
% Use new environment:
\begin{alphList}
  \item Hello.
  \item World.
\end{alphList}

```

---

‘List-’ followed by the current value of the counter as a lower case letter. Inside the second argument of the `list` environment, the command `\usecounter{ListCounter}` “uses” the counter, which basically adjusts the value of the counter for the next `\item`.

As part of  $\text{\LaTeX}$ ’s default mechanism all changes to counters and lengths inside an environment are local. This ensures that the counter `ListCounter` is reset to its original value upon leaving the environment.

Using the `list` environment over and over with the same arguments is not particularly useful and prone to errors. The `\newenvironment` command lets you define a new environment with an easier hassle-free interface. Figure 3.8 shows how you may implement the functionality of the previous example as a user-defined environment.

The first argument of `\newenvironment` is the name of the new environment. The second argument determines the commands which are carried out at the start of the environment. These are the commands which start the `list` environment. The last argument determines the commands which are carried out at the end on the environment. These

commands end the `list` environment. More information about `\newenvironment` may be found in Chapter 10.



# **Part III**

## **Pictures, Diagrams, Tables, and Graphs**





# CHAPTER 4

## Presenting External Pictures

THIS CHAPTER is an introduction to presenting pictures which are stored in external files. Historically, this was an important mechanism for importing pictures. Since pictures are usually included as numbered figures, this chapter also provides an introduction to the `figure` environment and, more generally, *floating* environments.

The remainder of this chapter, is mainly based on [D. P. Carlisle, 2003; Carlisle and Ratz, 1999; Reckdahl, 2006; Lamport, 1994]. It starts by introducing the `figure` environment and continues by explaining how to include external pictures. This chapter is included mainly for completeness as Chapter 5 is an introduction to specifying pictures and diagrams with the `tikz` package, which is built on top of the `pgf` package. Furthermore, Chapter 7 shows how to present graphs using the `pgfplots` package, which is also built on top of `pgf`. Readers not using external graphics are advised to only read the following two section and skip the remainder of this chapter.

### 4.1 The `figure` Environment

The `figure` environment is usually used to present pictures, diagrams, and graphs. The environment creates a *floating* environment. *Floating* environments don't allow pagebreaks and they may "float" to a convenient location in the output document [Lamport, 1994]. This mechanism gives  $\text{\LaTeX}$  more freedom to choose better page breaks for the remaining text. For example, if there's not enough room left for a figure at the "current" position then  $\text{\LaTeX}$  may fill up the remainder of the page with more paragraphs and put the figure on the next page. In this example the figure doesn't end up exactly where you intended it but the result is an aesthetically more pleasing document. However, it should be noted that you can also force the typesetting of a floating environment at the "current" position in the output file.

The body of a `figure` environment is typeset in a numbered figure. The `\caption` command may be used to define a caption of the figure.

$\text{\LaTeX}$  gives some control over the placement of floating figures, of floating tables, and other floats. For figures the placement is controlled with an optional argument of the `figure` environment. The same mecha-

nism is used for the `table` environment, which is explained in Chapter 6. The optional argument which controls the placement may contain any combination of the letters `'t'`, `'b'`, `'p'`, `'h'`, and `'H'`, which are used as follows [Lamport, 1994, Page 197]:

- `t` Put the float at the top of a page.
- `b` Put the float at the bottom of a page.
- `p` Put the float on a separate page with no text, but only figures, tables, and other floats.
- `h` Put the float approximately at the current position (here). (This option is not available for double-column figures and figures in two-column format.)
- `H` Put the float at the current position (Here). (This option is not available for double-column figures and figures in two-column format.)

The default value for the optional argument is `'tbp'`.  $\text{\LaTeX}$  parses the letters in the optional argument from left to right and will put the figure at the position corresponding to the first letter for which it thinks the position is “reasonable”. Good positions are the top of the page, the bottom of the page, or a page with floats only, as these positions do not disrupt the running text too much.

Inside the `figure` environment the command `\caption` defines a caption. The caption takes a *moving argument*, so fragile commands must be *protected*. Moving arguments and `\protect` are explained in Section 10.2.3. The regular argument defines the caption as it is printed in the figure and in the list of figures. If the regular argument gets too long then you may not want this text in the list of figures. In that case, you may add an optional argument, which is used to define a short alternative title for the list of figures. Within the regular argument of the `\caption` command, you may define a label for the figure with the `\label` command. This works as usual. The starred version of the environment (`figure*`) produces an unnumbered figure, which is not listed in the list of figures.

The following shows how to create a figure. Inside the `figure` environment you can put  $\text{\LaTeX}$  statements to produce the actual figure. In this example the text ‘Comparison of algorithms.’ appears in the list of tables and the text ‘Comparison. The dashed line ...’ is used for the caption.

```
\begin{figure}[tbp]
  (Insert \LaTeX here to produce the figure.)
  \caption[Comparison of algorithms.]
    {Comparison. The dashed line ...
      \label{fig:comparison}}
\end{figure}
```

$\text{\LaTeX}$  Usage

**Figure 4.1**  
Using the `dpfloat` package.

---

```

\begin{figure}[p]
% Left-side part of float(possibly with \caption).
\begin{leftfullpage}
  \langle Left part of float \rangle
\end{leftfullpage}
\end{figure}
% Right-side part of float(possibly with \caption).
\begin{figure}[p]
\begin{fullpage}
  \langle Right part of float \rangle
\end{fullpage}
\end{figure}

```

---

## 4.2 Special Packages

This section presents some packages which overcome some of  $\text{\LaTeX}$ 's limitations for floating environments.

### 4.2.1 Floats

$\text{\LaTeX}$  always forces the caption of floating environments on the same page as that of the environment. The `dpfloat` package lets you create floating environments on consecutive pages. This may be useful, for example, if your float is too large and you want the caption on the opposite page. Figure 4.1 presents an example.

### 4.2.2 Legends

Some documents distinguish between captions and *legends*. For such documents the caption of an environment consists of the name, the number, and a short title of the environment. For example, ‘Figure 4.1: Using the `dpfloat` package.’. The legend is a longer explanation of what’s in the environment.  $\text{\LaTeX}$  does not distinguish between captions and legends. The `ccaption` package overcomes this problem and provides a `\legend` command for legends. The package also provides support for caption placement and “sub-floats”.

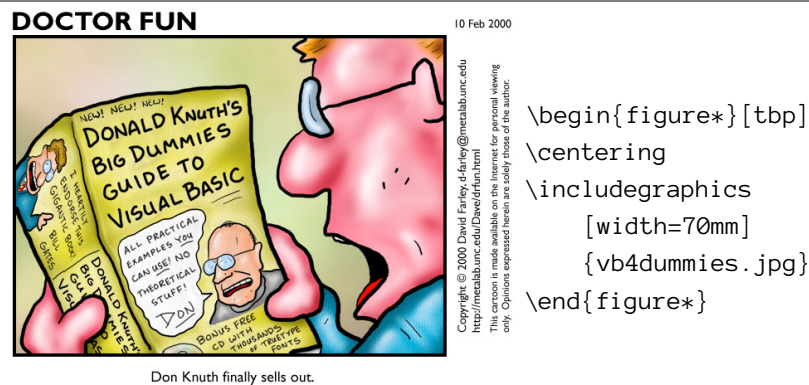
## 4.3 External Picture Files

A common mechanism for creating pictures is including them from external files. The best picture formats are vector graphics formats. The advantage of vector graphics is that they scale properly and always give the graphics a smooth appearance. Vector graphics formats which work well with  $\text{\LaTeX}$  are `.eps` and `.pdf`.

Programs such as `gnuplot` may be used to generate graphs in vector graphics format from your data. A common practice is generating complicated graphs with `gnuplot` and including them with  $\text{\LaTeX}$ . This mechanism is relatively easy. However, `gnuplot` may not always have the

Figure 4.2

Including an external graphics file with the `\includegraphics` command. The input to the right results in the output to the left. The Doctor Fun picture is included with the `\kind` permission from David Farley.



```
\begin{figure*}[tbp]
\centering
\includegraphics
[width=70mm]
{vb4dummies.jpg}
\end{figure*}
```

right graph output style. Another problem with externally generated pictures is that they may not always give a consistent look and feel as a result of differences in fonts and scaling. The `pgfplots` package overcomes these problems. (This package is explained in Chapter 7.)

## 4.4 The `graphicx` Package

The `graphicx` package provides a command called `\includegraphics` which supports the inclusion of external graphics in an easy way.

`\includegraphics[⟨key-value list⟩]{⟨file⟩}`

This includes the external graphics file `⟨file⟩`. The optional argument is a `⟨key⟩=⟨value⟩` list controlling the scale, size, rotation, and other aspects of the picture. The following describes some of the possible keys. Information about other `⟨key⟩=⟨value⟩` combinations may be found in the `graphicx` package documentation [Carlisle and Ratz, 1999].

**angle** The the rotation angle in degrees.

**width** The width of the resulting picture. The width should be specified in a proper dimension, e.g., 5cm, 65mm, 3in, and so on. The height of the picture is scaled to match the given width.

**height** The height of the resulting picture. This is the dual of the width key.

**type** Specifies the file type. The file type is normally determined from the filename extension. □

Figure 4.2 shows an example of the `\includegraphics` command. In this example, the command is used in the body of a `figure*`, which is the unnumbered version of a figure. The picture is a reproduction from the Dr Fun pages (<http://www.ibiblio.org>).

## 4.5 Setting Default Key Values

The `graphicx` package uses the `keyval` package to handle its `⟨key⟩=⟨value⟩` pairs. The `keyval` package lets you define a default value for

each key. The following is a short explanation. A full explanation may be found Section 11.2. Basically, the command `\setkeys{Gin}{⟨list⟩}` sets the defaults. Here `⟨list⟩` is a comma-separated `⟨key⟩=⟨value⟩` list. The following is an example, which sets the default width to 6 cm.

```
\setkeys{Gin}{width=6cm}
```

*LaTeX Usage*

After specifying this command there is no more need to specify the width of the pictures: the width is now 6 cm by default. However, it is still possible to override this default width by providing an explicit width.

Sometimes it is nicer to specify a width and/or height as a fraction of the current page dimensions. This may be done as follows:

```
\setkeys{Gin}{width=0.9\textwidth,height=0.9\textheight}
```

*LaTeX Usage*

## 4.6 Setting a Search Path

By default `\includegraphics` searches the current directory for files. However, it is also possible to define a search path. The search path mechanism works similar to a Unix search path. The command `\graphicspath{⟨directory list⟩}` sets the search path to `⟨directory list⟩`, which consists of a list of directories, each of which should be inside a brace pair. The following is an example which sets the search path to `‘./pdf/, ./eps’`. Notice the absence of commas in the list.

```
\graphicspath{{./pdf/}{./eps/}}
```

*LaTeX Usage*

## 4.7 Defining Graphics Extensions

The kind of graphics extensions allowed by `\includegraphics` depends on the extension of your output file. The last argument of `\includegraphics` determines the name of the external graphics file. It is allowed to omit the file extension. When `\includegraphics` sees a filename without extension it will try to add a proper extension. The command `\DeclareGraphicsExtensions{⟨extension list⟩}` lets you specify the allowed file extensions which may be added to filenames without extensions. The argument `⟨extension list⟩` is a comma-separated list of extensions. The command works as expected. If an extension is omitted in the required argument of the `\includegraphics` command, the `⟨extension list⟩` is searched from left to right. The process halts when an extension is found which “completes” the partial filename. The partial filename and the extension are used as the external graphics filename. You may disallow filenames without extensions by applying the command `\DeclareGraphicsExtensions{}`.

## 4.8 Conversion Tools

This section briefly discusses some approaches to the conversion of graphics formats.

If you only have a few pictures to convert and you've never used a command-line tool to convert pictures then you're probably best off converting your pictures with a program with a graphical user interface. The program `gimp`, which comes with most modern Unix flavours, is free and is pretty easy to use. It supports the conversion from and to several graphics formats.

However, if you have to convert many pictures then you may be better off with a command-line tool. The following are some available tools.

**epstopdf** Converts from `.eps` to `.pdf`. You can also use this program to convert postscript output from `gnuplot`.

**gs** This is the GhostScript conversion program, which is capable of conversions to and from more than 300 different graphics formats. The following is a shell script which uses `gs` to convert `.eps` to `.pdf`. The example is easily modified for other conversions to `.pdf`.

```
# Convert from eps to pdf.
GS=/usr/bin/gs
BASENAME='basename $1 .eps'
${GS} -sDEVICE=pdfwrite -dNOPAUSE -dQUIET \
      -sOutputFile=${BASENAME}.pdf - < $1
```

Unix Script

## 4.9 Defining Graphics Conversion

This section briefly mentions the notion of *graphics conversion rules*, which are used to automate the conversion of graphics format files from within  $\text{\LaTeX}$ . These rules work in combination with `\includegraphics`. The actual conversion is specified with the `\DeclareGraphicsRule` command, which is not easy to use. You are advised to stick to the allowed graphics extensions and convert non-allowed formats to allowed formats by hand, choosing vector graphics if possible. If you use `pdf $\text{\LaTeX}$`  then the following extensions are supported: `.png`, `.pdf`, `.jpg`, and `.mps`. Here the `.mps` format is for METAPOST source files, which are translated to encapsulated postscript with the aid of the program `mpost`. Further information about the `\DeclareGraphicsRule` command may be found in [Reckdahl, 2006, Section 9.2] or in the graphics package documentation [D. P. Carlisle, 2003].

# CHAPTER 5

## Presenting Diagrams with `tikz`

THIS CHAPTER is an introduction to drawing diagrams/pictures using the `tikz` package, which is built on top of `pgf`. Here `pgf` is a platform- and format-independent macro package for creating graphics. The `pgf` package is smoothly integrated with  $\text{\TeX}$  and  $\text{\LaTeX}$ . As a result `tikz` also lets you incorporate text and mathematics in your diagrams. The `tikz` package also supports the `beamer` package, which is used for creating incremental presentations. (Chapter 14 is an introduction to `beamer`.)

The main purpose of this chapter is to whet the appetite. The presentation is mainly based on CVS version 2.00 (CVS2010-01-03) of `pgf` and `tikz`. The interested reader is referred to the excellent package documentation [Tantau, 2010] for more detailed information.

This chapter starts with a section which discusses the advantages and disadvantages of specifying diagrams. This is followed by a quick introduction to the `tikzpicture` environment and some drawing commands. Next there is a crash course on some of the more common and useful `tikz` commands. Finally, there are introductions to some of the `tikz` libraries. By the end of this chapter you should know how to draw maintainable, high-quality graphics consisting of basic shapes such as points, lines, and circles, but also of trees, finite state automata, entity-relationship diagrams, and neural networks.


### 5.1 Why Specify your Diagrams?

### 5.2 The `tikzpicture` Environment

The `tikz` package — `tikz` is an acronym of ‘`tikz` ist kein Zeichenprogramm’ — provides commands and environments which let you specify and ‘draw’ graphical objects in your document. The package is smoothly integrated with  $\text{\TeX}$  and  $\text{\LaTeX}$ , so graphical objects also can be text. What is more, the things you specify/draw may have attributes. For example, tree nodes have coordinates and may have parts such as children, grandchildren, and so on. The package also supports mathematical and object oriented computations.

Drawing with `tikz` may be done in different ways, but to simplify matters we shall do most of our drawing inside a `tikzpicture` environ-

ment. Each `tikzpicture` results in a box. The size of the box is the smallest possible box containing material which is typeset in the box. Only the relative positions of the coordinates inside a `tikzpicture` matter. For example, a `tikzpicture` consisting of a  $2 \times 2$  square which is drawn at coordinate  $(1, 2)$  in the `tikzpicture` results in the same graphic on your page as a `tikzpicture` consisting of a  $2 \times 2$  square which is drawn at coordinate  $(0, 0)$  in your `tikzpicture`. All *implicit* units inside a `tikzpicture` are in centimetres. Scaling a `tikzpicture` is done by specifying an optional ‘scale = number’ argument which is passed to the `tikzpicture` environment. This kind of scaling only applies to the actual coordinates but *not* to line thicknesses, font sizes, and so on. This makes sense, as you would not want, say, the font in your diagrams to be of a different kind than the font in your running text. The package also supports other top-level options.

The following draws a  $0.5 \times 0.25$  crossed rectangle: .

L<sup>A</sup>T<sub>E</sub>X Input

```
The following draws
a $0.5 \times 0.25$ crossed rectangle:
\begin{tikzpicture}
\draw (0.00,0.00) rectangle (0.50,0.25);
\draw (0.00,0.00) -- (0.50,0.25);
\draw (0.00,0.25) -- (0.50,0.00);
\end{tikzpicture}\,.
```

Of course the previous example violates almost every rule in the maintainability book. For example, what if the size of the rectangle were to change, what if the position were to change, what if the colour were to change, ...?

Fortunately, `tikz` provides a whole arsenal of commands and techniques which help you keep your diagrams maintainable. One of the cornerstones is the ability to label *nodes* and *coordinates* — a special kind of nodes — in your `tikzpicture`. Once you’ve defined your labels you can use them to construct other nodes and shapes. In addition the package supports hierarchies. Parent settings may be inherited by descendants in the hierarchy. The second and third next sections explain how to construct paths, and how to use nodes, coordinates, and labels. Before we start with them, we shall study the `\tikz` command and how to draw grids.

## 5.3 The `\tikz` Command

Using the `tikzpicture` environment for small in-line diagrams which only require a simple command is time and space consuming. Fortunately, `tikz` also defines the following command.

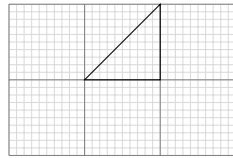
```
\tikz[⟨options⟩]{⟨commands⟩}
```

This works similar to ‘`\begin{tikzpicture}[⟨options⟩]⟨commands⟩\end{tikzpicture}`’.

□



**Figure 5.1**  
Drawing a grid.




---

```
\begin{tikzpicture}
\draw[line width=0.1pt,gray!30,step=1mm]
  (0,0) grid (3,2);
\draw[help lines]
  (0,0) grid (3,2);
\draw (1,1) -- (2,2) -- (2,1) -- cycle;
\end{tikzpicture}
```

---

`\tikz[⟨options⟩] ⟨command⟩;`

If `⟨command⟩` is a single command then this is equivalent to `'\tikz[⟨options⟩]{⟨command⟩;}'`. □

## 5.4 Grids

Drawing a grid is a useful technique which allows us to quickly relate the positions of what's in your picture. Grids are also useful when you are developing a picture. The following shows two ways to draw a grid. The former way is easier, but it is expressed in terms of the second, more general, notation.

`\draw[⟨options⟩] ⟨start coordinate⟩ grid ⟨end coordinate⟩;`

This draws a grid from `⟨start coordinate⟩` to `⟨end coordinate⟩`. The optional argument may be used to control the style of the grid.

The option `'step = ⟨dimension⟩'` is used for setting the distance between the lines in the grid. There are also directional versions `'xstep = ⟨dimension⟩'` and `'ystep = ⟨dimension⟩'` for setting the distances in the *x*- and *y*-directions. □

`\path[⟨path options⟩] ... grid[⟨options⟩] ⟨coordinate⟩ ... ;`

This adds a grid to the current path from the current position in the path to `⟨coordinate⟩`. To *draw* the grid, the option `draw` is required as part of `⟨path options⟩`. □

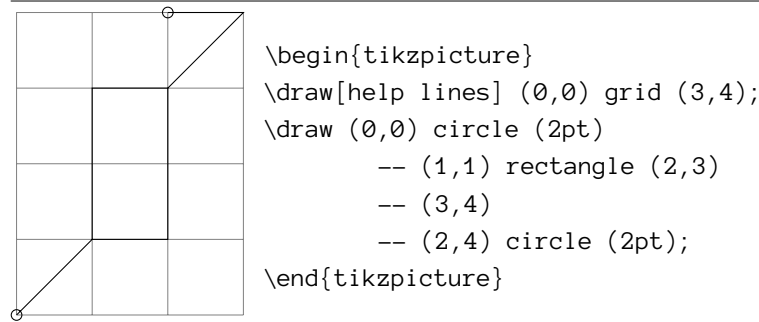
Figure 5.1 demonstrates how to draw a basic  $3 \times 2$  grid, relative to the origin. The grid consists of two superimposed grids, the coarser of which is drawn on top of the other. The option `'gray!20'` in the style of the fine grid defines the colour for the grid: you get it by mixing 20% grey and 80% white.

The option, style really, `'help lines'` is very useful because it results in lines drawn in a subdued colour. For the printable version of this book the style is redefined to make the lines very thin and to set the color to a combination of 50% black and 50% white. This was done with the command `\tikzset{help lines/.style={very thin,color=black!50}}`. Styles are explained in Section 5.13.

## 5.5 Paths

Inside a `tikzpicture` environment everything is drawn by starting a *path* and by *extending* the path. Paths are constructed using the `\path`

**Figure 5.2**  
Creating a path.



command. In its basic form, a path is started with a coordinate which becomes the *current* coordinate of the path. Next the path is extended with other coordinates, line segments, *nodes* or other shapes. Line segments may be straight line segments or *cubic spline segments*, which are also known as *cubic splines*. Cubic splines are explained in Section 5.7. Each line segment extension operation adds a line segment starting at the current coordinate and ending at another coordinate. Path extension operations may update the current coordinate.

The optional argument of the `\path` command is used to control if, and how the path should be drawn. Adding the option ‘draw’ forces the drawing of the path. By default the path is *not* drawn. A semicolon indicates the end of the path:

```

\begin{tikzpicture}
\path[draw] (1,0) -- (2,0);
\path      (0,0) -- (3,0);
\end{tikzpicture}

```

*LaTeX Input*

The first `\path` command in this `tikzpicture` draws a line segment from  $(1,0)$  to  $(2,0)$ . The second `\path` command results in a line segment which is not drawn. Still the line segment is considered part of the picture, so the picture has a width of 3 cm.

The command `\draw` is a shorthand for `\path[draw]`. The `tikz` package has many shorthand notations like this.

Figure 5.2 demonstrates the drawing of a path which starts at position  $(0,0)$ . The path is extended by adding a circle, is extended with a line segment to  $(1,2)$ , is extended with a rectangle, and so on. Except for the ‘circle’ extension operation, each operation changes the current position of the path.

## 5.6 Coordinate Labels

Maintaining complex diagrams defined entirely in terms of absolute coordinates is virtually impossible. Fortunately, `tikz` provides many techniques which help you maintain your diagrams. One of these techniques is the ability to define *coordinate labels* and use the resulting labels instead of the coordinates.

You define a coordinate label by writing ‘`coordinate(<label>)`’ after

the coordinate. Defining coordinates this way is possible at (almost) any point in a path. Once the label of a coordinate is defined, you can use ‘ $\langle \text{label} \rangle$ ’ as if it were the coordinate. The following, which draws a crossed rectangle ( $\boxtimes$ ), demonstrates the mechanism. It is not intended to excel in terms of maintainability.

The following, which draws a crossed rectangle

```
(\begin{tikzpicture}
  \draw (0.0,0.0) coordinate(lower left)
        -- (0.4,0.2) coordinate(upper right);
  \draw (0.0,0.2) -- (0.4,0.0);
  \draw (lower left) rectangle (upper right);
\end{tikzpicture}), demonstrates the mechanism.
```

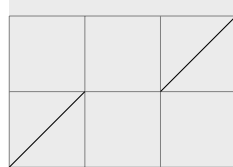
LaTeX Input

## 5.7 Extending Paths

As explained before, paths are constructed by *extending* them. There are several different kinds of path extension operations. The majority of these extension operations modify the current coordinate, but some don't. In the remainder of this section it is therefore assumed that an extension operation modifies the current coordinate *unless* this is indicated otherwise. For the moment it is assumed that none of the coordinates are *relative* or *incremental coordinates*, which are explained in Section 5.11.1. The following are the common extension operations.

`\path ...  $\langle \text{coordinate} \rangle$  ...;`

This is the *move-to* operation, which adds the coordinate  $\langle \text{coordinate} \rangle$  to the path. The following example uses three move-to operations. The first move-to operation defines the lower left corner of the grid. The remaining move-to operations define the starts of two line segments.

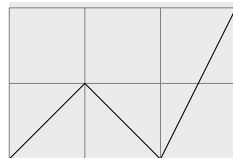


```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1)
      (2,1) -- (3,2);
\end{tikzpicture}
```



`\path ... --  $\langle \text{coordinate} \rangle$  ...;`

This is the *line-to* operation, which adds a straight line segment to the path. The line segment is from the current coordinate and ends in  $\langle \text{coordinate} \rangle$ . The following is an example.



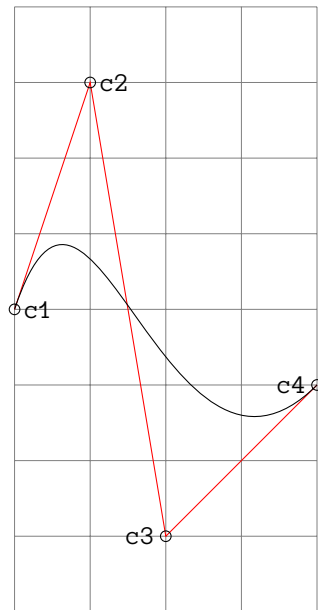
```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (0,0) -- (1,1) -- (2,0) -- (3,2);
\end{tikzpicture}
```



`\path ... .. controls  $\langle \text{coordinate}_1 \rangle$  and  $\langle \text{coordinate}_2 \rangle$  ..  $\langle \text{coordinate}_3 \rangle$  ...;`

This is the *curve-to* operation, which adds a *cubic Bézier spline segment* to the path. The start point of the curve is the current point of the path. The end point is  $\langle \text{coordinate}_3 \rangle$ , and the control points are  $\langle \text{coordinate}_1 \rangle$

**Figure 5.3**  
Cubic spline in tikz.



```
\begin{tikzpicture}
\draw[help lines] (-2,-4) grid (+2,+4);
\path (-2,+0) coordinate(c1)
      (-1,+3) coordinate(c2)
      (+0,-3) coordinate(c3)
      (+2,-1) coordinate(c4);
\draw[red] (c1) -- (c2) -- (c3) -- (c4);
\draw (c1) circle (2pt)
      (c2) circle (2pt)
      (c3) circle (2pt)
      (c4) circle (2pt)
      (c1) .. controls (c2)
              and (c3) .. (c4)
      (c1) node[anchor=west] {\texttt{c1}}
      (c2) node[anchor=west] {\texttt{c2}}
      (c3) node[anchor=east] {\texttt{c3}}
      (c4) node[anchor=east] {\texttt{c4}};
\end{tikzpicture}
```

and  $\langle \text{coordinate}_2 \rangle$ .

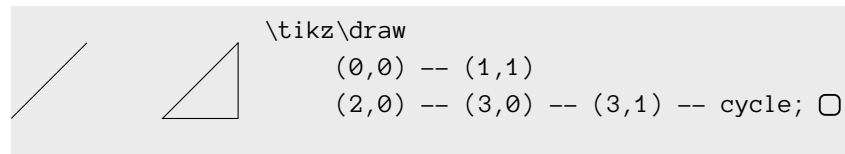
Figure 5.3 demonstrates the operation. The curve starts at  $c_1$  and ends at  $c_4$ . Its control points are given by  $c_2$  and  $c_3$ . The tangent of the spline segment at  $c_1$  is given by the tangent of the line segment ' $c_1$  --  $c_2$ '. Likewise, the tangent of the spline segment at  $c_4$  is given by the tangent of the line segment ' $c_3$  --  $c_4$ '. This makes cubic Bézier splines a perfect candidate for approximating complex curves as a sequence of spline segments. By properly choosing the start point, the end point, and the control points of the segments, you can enforce continuity both in the curves *and* the first derivative. (As a matter of fact, it is also possible to ensure continuity in the second derivative.) Notice that the start, end, and control points need not be equidistant, nor need the start and end point lie on a horizontal line.  $\square$

```
\path ... .. controls  $\langle \text{coordinate}_1 \rangle$  ..  $\langle \text{coordinate}_2 \rangle$  ...;
```

This is also a curve-to operation. It is equivalent to the operation ' $\dots .. controls \langle \text{coordinate}_1 \rangle$  and  $\langle \text{coordinate}_1 \rangle .. \langle \text{coordinate}_2 \rangle \dots$ '.  $\square$

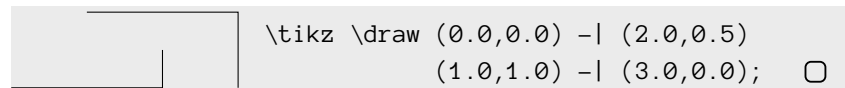
```
\path ... -- cycle ...;
```

This is the *cycle* operation which *closes* the current path by adding a straight line segment from the current point to the last destination point of a move-to operation. The cycle operation has three applications. First it *closes* the path. Closing a path is required if you wish to *fill* the path with a colour. Second, it properly connects the start and end line segments in the path. Third, it increases maintainability as it avoids referencing the start point of the path. The following is an example.



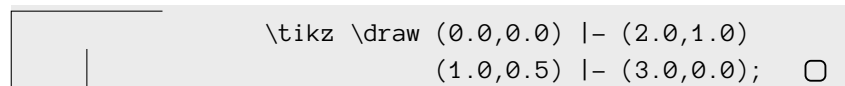
`\path ... -| <coordinate> ...;`

This operation is equivalent to two line-to operations connecting the current coordinate and `<coordinate>`. The first operation adds a horizontal and the second a vertical line segment. The following is an example.



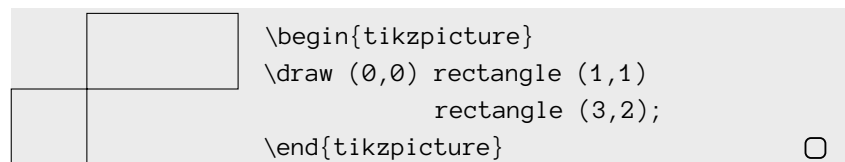
`\path ... |- <coordinate> ...;`

This operation is also equivalent to two line-to operations connecting the current coordinate and `<coordinate>`. This time, however, the first operation adds a vertical and the second a horizontal line segment. The following is an example.



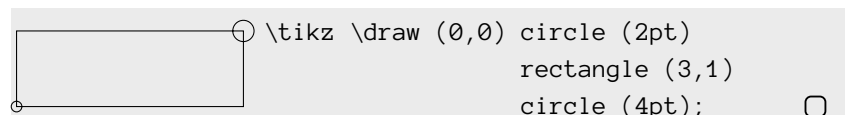
`\path ... rectangle <coordinate> ...;`

This is the *rectangle* operation, which adds a rectangle to the path. The rectangle is constructed by making the current coordinate and `<coordinate>` the lower left and upper right corners of the rectangle. Which coordinate determines which corner depends on the values of the coordinates. The following is an example.



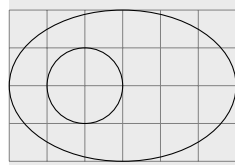
`\path ... circle (<radius>) ...;`

This is the *circle* operation, which adds a circle to the path. The centre of the circle is given by the current coordinate of the path and its radius is the dimension `<radius>`. This operation does *not* change the current coordinate of the path. The following is an example.



`\path ... ellipse(<half width> and <half height>) ...;`

This is the *ellipse* operation, which adds an ellipse to the path. The centre of the ellipse is given by the current coordinate. This operation does *not* change the current coordinate of the path. The following is an example.

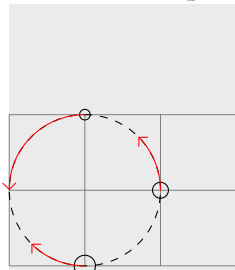


```
\begin{tikzpicture}[scale=0.5]
\draw[help lines] (0,0) grid (6,4);
\draw (2,2) ellipse (1cm and 1cm)
      (3,2) ellipse (3cm and 2cm);
\end{tikzpicture}
```

□

`\path ... arc (<start angle>:<end angle>:<radius>) ...;`

This is the *arc* operation, which adds an arc to the path. The arc starts at the current point. The radius is given by the dimension `<radius>`. The start and end angles of the arc are given by `<start angle>` and `<end angle>` respectively. This operation does *not* change the current coordinate of the path. The following is an example. In this example, the draw option `‘->’` tells the draw to draw the path as an arrow. The `tikzpicture` option `‘>=angle 90’` sets the style of the arrow head.



```
\begin{tikzpicture}[>=angle 90]
\draw[help lines] (0,0) grid (3,2);
\draw[dashed] (1,1) circle (1cm);
\draw (1,2) coordinate(a) circle (2pt)
      (2,1) coordinate(b) circle (3pt)
      (1,0) coordinate(c) circle (4pt);
\draw[>,red] (a) arc (90:180:1cm);
\draw[>,red] (b) arc (0:45:1cm);
\draw[>,red] (c) arc (270:225:1cm);
\end{tikzpicture}
```

□

`\path ... arc (<start angle>:<end angle>:<radius> and <half height>) ...;`

This is also an *arc* operation but this time it adds a segment of an ellipse to the path.

□

## 5.8 Actions on Paths

So far most of our examples have used the default path style. This may not always be what you want. For example, you may want to draw a line in a certain colour, change “its” width, fill a shape with a colour, and so on. In `tikz` terminology you achieve this with *path actions*, which are operations on an existing path. You first construct the path and then apply the action. At the basic level the command `\draw` is defined in terms of an action on a path: the action results in the path being drawn. As pointed out before `\draw` is a shorthand for `\path[draw]`.

The following are some other shorthand commands which are defined in terms of path actions inside the `tikzpicture` environment.

`\draw`

This is a shorthand for `\path[draw]`.

```
\tikz
\draw (0,0) -- (3,0);
```


□

`\fill`

This is a shorthand for `\path[fill]`.


**Table 5.1**  
The xcolor colours.

 black	 darkgray	 lime	 pink	 violet
 blue	 gray	 magenta	 purple	 white
 brown	 green	 olive	 red	 yellow
 cyan	 lightgray	 orange	 teal	

 \tikz  
\fill[lime] (0,0) rectangle (3,0.5); ☐


\filldraw

This is a shorthand for \path[filldraw].

 \tikz  
\filldraw[fill=lime,draw=red]  
(0,0) rectangle (3,0.5); ☐

\shade


This is a shorthand for \path[shade].

 \tikz  
\shade[left color=lime,right color=teal]  
(0,0) rectangle (3,0.5);

Shading paths is possible in many ways. The reader is invited to read the pgf manual [Tantau, 2010] for further information. ☐

\shadedraw

This is a shorthand for \path[shadedraw].

 \tikz  
\shadedraw[left color=lime,  
right color=teal,  
draw=red]  
(0,0) rectangle (3,0.5); ☐

### 5.8.1 Colour

The tikz package is aware of several built-in colours. Some of these colours are inherited from the xcolor package [Kern, 2007]. Table 5.1 depicts some of them.

#### Defining New Colours

There are several techniques to define a new name for a colour.

\definecolor{<name>}{rgb}{<red ratio>,<green ratio>,<blue ratio>}

This defines a new colour called <name> in the ‘rgb’ model. The colour is the result of combining <red ratio> parts red, <green ratio> parts green, and <blue ratio> parts blue. All ratios should be reals in the interval [0 : 1]. ☐

\definecolor{<name>}{gray}{<ratio>}

This defines a new colour called <name> which has a <ratio> grey part

in the ‘gray’ model. The value of  $\langle \text{ratio} \rangle$  should be a real in the interval  $[0 : 1]$ . □

```
\colorlet{<name>}{<colour>!<percentage>}
```

This defines a new colour called  $\langle \text{name} \rangle$  which is the result of mixing  $\langle \text{percentage} \rangle\%$   $\langle \text{colour} \rangle$  and  $(100 - \langle \text{percentage} \rangle)\%$  white. Here  $\langle \text{colour} \rangle$  should be the name of an existing colour. □

```
\colorlet{<name>}{<colour_1>!<percentage>!<colour_2>}
```

This defines a new colour called  $\langle \text{name} \rangle$  which is the result of mixing  $\langle \text{percentage} \rangle\%$   $\langle \text{colour}_1 \rangle$  and  $(100 - \langle \text{percentage} \rangle)\%$   $\langle \text{colour}_2 \rangle$ . Here  $\langle \text{colour}_1 \rangle$  and  $\langle \text{colour}_2 \rangle$  should be names of existing colours.

Other, more exotic expressions are also allowed. More information about the allowed colour mixing expressions may be found in the documentation of the `xcolor` package [Kern, 2007]. □

### Using the Colour

Some path actions let you set the colour for the operation. For example, you may draw a path with the given colour. There are different ways to control the colour. The *option* ‘color’ determines the colour for drawing and filling. It also sets the colour of the text in nodes. (Nodes are explained in Section 5.9.

You may set the colour of the whole `tikzpicture` or set the colour of a given path action. Setting the colour of the whole picture is done by passing a ‘color= $\langle \text{colour} \rangle$ ’ option to the environment. Setting the colour of a path action is done by passing the option to the `\path` command (or its derived shorthand commands). The following is an example which draws four lines: one in blue, one in red, one in 20% red and 80% white, and one in 40% red and 60% blue.



```
\begin{tikzpicture}[color=blue]
\draw (0,0) -- (2,0);
\draw[color=red] (0,1) -- (2,1);
\draw[color=red!20] (0,2) -- (2,2);
\draw[color=red!40!blue] (0,3) -- (2,3);
\end{tikzpicture}
```

The `tikzpicture` environment and the `\path` command (and derived commands) are pretty relaxed about colours and let you omit the ‘color=’ part when specifying the colour option. The following is perfectly valid.



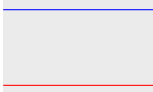
```
\begin{tikzpicture}[red]
\draw (0,0) -- (2,0);
\draw[color=blue] (0,1) -- (2,1);
\draw[red!20] (0,2) -- (2,2);
\end{tikzpicture}
```



### 5.8.2 Drawing the Path

As already mentioned, the `draw` option forces the drawing of a path. By specifying a `'draw = <colour>'` option the path will be drawn with the colour `<colour>`. Note that setting the `draw` option overrides the `colour` option.

The following demonstrates the mechanism. This example draws two lines. One is drawn in red. The other is drawn in blue.




```
\begin{tikzpicture}[red]
\draw      (0,0) -- (2,0);
\draw[draw=blue] (0,1) -- (2,1);
\end{tikzpicture}
```

### 5.8.3 Line Width

There are several path actions affecting the “line” style, i.e. the style that determines the line width, the line cap, and the line join. The following are some commands which affect the line width.

`line width=<dimension>`

This sets the line width to `<dimension>`.




```
\tikz \draw[line width=8pt]
      (0,0) -- (2,4pt);
```

☐

ultra thin

This sets the line width to 0.1 pt.

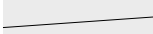


```
\tikz \draw[ultra thin]
      (0,0) -- (2,4pt);
```

☐

very thin

This sets the line width to 0.2 pt.




```
\tikz \draw[very thin]
      (0,0) -- (2,4pt);
```

☐

thin

This sets the line width to 0.4 pt.




```
\tikz \draw[thin]
      (0,0) -- (2,4pt);
```

☐

semithick

This sets the line width to 0.6 pt.

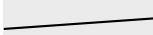


```
\tikz \draw[semithick]
      (0,0) -- (2,4pt);
```

☐

thick

This sets the line width to 0.8 pt.

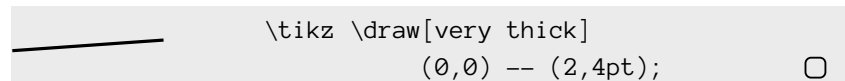


```
\tikz \draw[thick]
      (0,0) -- (2,4pt);
```

☐

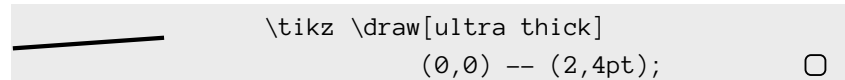
very thick

This sets the line width to 1.2 pt.



ultra thick

This sets the line width to 1.6 pt.

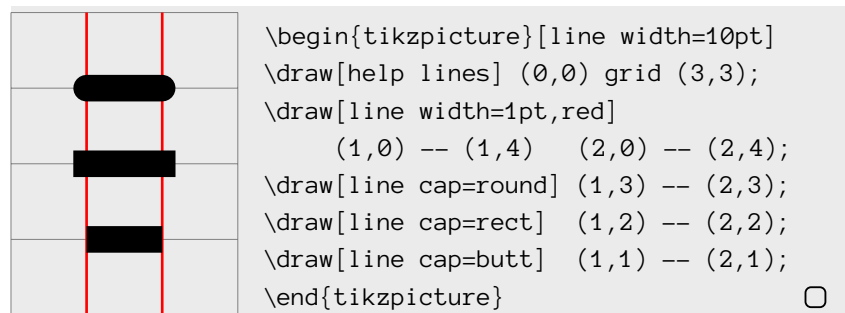


### 5.8.4 Line Cap and Join

The drawing of a path depends on several parameters. The *line cap* determines how lines start and end. The *line join* determines how line segments are joined.

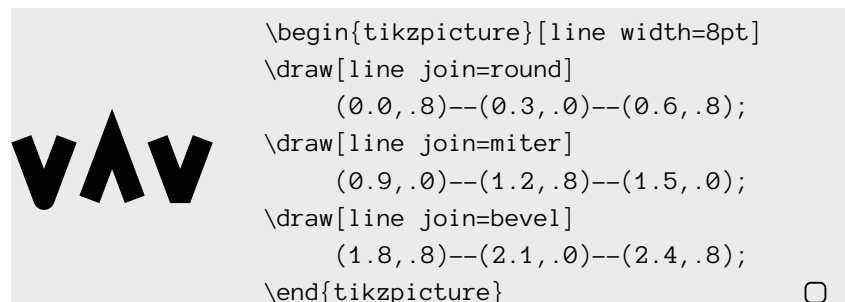
`line cap=<style>`

This sets the line cap style to `<style>`. There are three possible values for `<style>`: 'round', 'rect', and 'butt'.



`line join=<style>`

This sets the line join style to `<style>`. There are three possible values for `<style>`: 'round', 'miter', and 'bevel'.



`miter limit=<fraction>`

This option avoids miter joins with sharp angles which protrude too far beyond the joining point. This works by specifying a limit on how far the miter join is allowed to protrude the joining point. If the join protrudes beyond this limit then the join style is changed to bevel. The actual value of the limit is given by the product of `<fraction>` and the line width.

**Figure 5.4**  
Using a dash pattern

```
\begin{tikzpicture}
\draw[dash pattern=on 4mm off 1mm on 4mm off 2mm]
  (0,0.5) -- (2,0.5);
\draw[dash pattern=on 3mm off 2mm on 3mm off 3mm]
  (0,0.0) -- (2,0.0);
\end{tikzpicture}
```

**Figure 5.5**  
Using a dash phase.

```
\begin{tikzpicture}[dash pattern=on 3mm off 2mm]
\draw[dash phase=3mm] (0,0.5) -- (2,0.5);
\draw[dash phase=2mm] (0,0.0) -- (2,0.0);
\end{tikzpicture}
```



```
\begin{tikzpicture}
[line width=8pt,line join=miter]
\draw (0,0) -- (0.25,2) -- (0.5,0);
\draw[miter limit=8]
  (1,0) -- (1.25,2) -- (1.5,0);
\end{tikzpicture}
```

### 5.8.5 Dash Patterns

The drawing of lines also depends on the *dash pattern* setting. By default it is *solid*. The following shows the relevant path actions which affect dash patterns.

`dash pattern=<pattern>`

This sets the dash pattern to `<pattern>`. The syntax for `<pattern>` is the same as in METAFONT. Basically `<pattern>` specifies a cyclic pattern of lengths which determine when the line should be drawn (when it's on) and when it should not be drawn (when it's off). You usually write the lengths in terms of multiples of points (pt). Figure 5.4 presents an example which uses millimetres for simplicity. □

`dash phase=<dimension>`

This shifts the dash phase by `<dimension>`. Figure 5.5 shows an example. □

`solid`

This is the default dash pattern style: it produces a solid line.

```
\tikz \draw[solid] (0,0) -- (2,0);
```

`dotted`

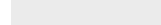
This is a predefined dash pattern style which produces a dotted line.

```
\tikz \draw[dotted] (0,0) -- (2,0);
```

`densely dotted`

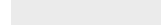
This is a predefined dash pattern style which produces a densely dotted line.

loosely dotted



```
\tikz
\draw[densely dotted] (0,0) -- (2,0); \quad \square
```

This is a predefined dash pattern style which produces a loosely dotted line.



```
\tikz
\draw[loosely dotted] (0,0) -- (2,0); \quad \square
```

dashed

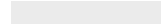
This is a predefined dash pattern style which produces a dashed line.



```
\tikz \draw[dashed] (0,0) -- (2,0); \quad \square
```

densely dashed

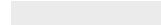
This is a predefined dash pattern style which produces a densely dashed line.



```
\tikz
\draw[densely dashed] (0,0) -- (2,0); \quad \square
```

loosely dashed

This is a predefined dash pattern style which produces a loosely dashed line.



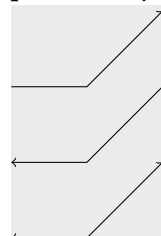
```
\tikz
\draw[loosely dashed] (0,0) -- (2,0); \quad \square
```

### 5.8.6 Arrows

Arrows are also drawn using path actions.

`arrows=<arrow head1>-<arrow head2>`

This adds an arrow head to the start and to the end of the path. You may also omit the ‘arrows=’ and use the shorthand notation ‘<arrow head<sub>1</sub>>-<arrow head<sub>2</sub>>’. The arrow head at the start is determined by <arrow head<sub>1</sub>>. The arrow head at the end is determined by <arrow head<sub>2</sub>>. Omitting <arrow head<sub>1</sub>> omits the arrow head at the start of the path. Omitting <arrow head<sub>2</sub>> omits the arrow head at the end. The following example demonstrates the mechanism for the default arrow head types ‘<’ and ‘>’. Table 5.2 list some of the available arrow head styles, some of which are provided by the `tikz` library `arrows`.






















```
\begin{tikzpicture}
\draw[->] (0,2) -- (1,2) -- (2,3);
\draw[<-] (0,1) -- (1,1) -- (2,2);
\draw[<->] (0,0) -- (1,0) -- (2,1);
\end{tikzpicture} \quad \square
```

`>=<end arrow type>`

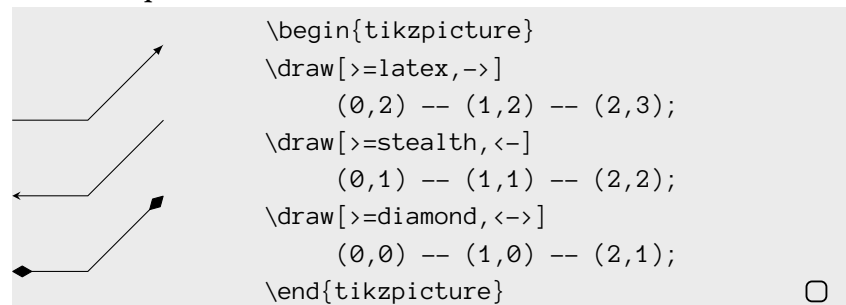
This redefines the *default* end arrow head style ‘>’. As already mentioned, some existing arrow head styles are listed in Table 5.2. Some of these arrow head types are provided by the `tikz` library `arrows`. Most styles

**Table 5.2**

Some available arrow head types. The arrows in the upper part of the table are predefined. The arrows in the lower part of the table are provided by the `tikz` library `arrows`.

				Predefined	
Style	Arrow	Style	Arrow	Style	Arrow
stealth		to		latex	
space					
Provided by arrows					
open triangle 90		triangle 90		angle 90	
open triangle 60		triangle 60		angle 60	
open triangle 45		triangle 45		angle 45	
open diamond		diamond		o	
open square		square		*	

in the table also have a “reversed style”, for example ‘`latex reversed`’, which just changes the direction of the ‘`latex`’ arrow head. The library may be loaded with the command `\usetikzlibrary{arrows}`, which should be in the preamble of your document. The following provides a small example.



### 5.8.7 Filling a Path

Not only can you draw paths but also can you fill them or draw them with one colour and fill them with a different colour. The only requirement is that the path be closed. Closing a path is done with the ‘`cycle`’ annotation. The following are the relevant commands.


`\path[fill=<colour>] <paths>;`

This fills each path in `<paths>` with the colour `<colour>`. Unclosed paths are closed first. It is also allowed to use ‘`color=<colour>`’. Finally, the option ‘`fill`’ on its own fills the paths with the last defined value for `fill` or for `color`.



`\fill[⟨options⟩] ⟨paths⟩;`

The command `\fill` on its own works just as `\path[fill=⟨colour⟩]`, where `⟨colour⟩` is the last defined value for `fill` or for `color`. Using `\fill` with options works as expected. The options are passed to `\path` and the paths in `⟨paths⟩` are filled.



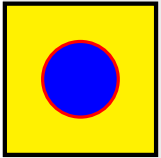
```

\begin{tikzpicture}[scale=0.4,fill=cyan]
\fill[color=teal]
  (0,0) -- (1,0) -- (1,1);
\fill[fill=green]
  (0,1) -- (0,2) -- (1,2) -- cycle;
\fill[black]
  (2,0) -- (3,0) -- (3,1) -- cycle;
\fill(2,1) -- (2,2) -- (3,2);
\end{tikzpicture}

```

`\filldraw[options] ⟨paths⟩;`

The command `\filldraw` fills and draws the path. The style ‘draw’ determines the drawing colour and the style ‘fill’ determines the filling colour. Both styles may be set in the optional argument. The following is an example.



```

\begin{tikzpicture}
\filldraw[ultra thick,fill=yellow]
  (0,0) rectangle (2,2);
\filldraw[very thick,fill=blue,draw=red]
  (1,1) circle (0.5cm);
\end{tikzpicture}

```

### 5.8.8 Path Filling Rules

There are two options which control how overlapping paths are filled. These rules determine which points are inside the shape. By cleverly using these options and by making paths overlap properly you can construct “holes” in the filled areas.

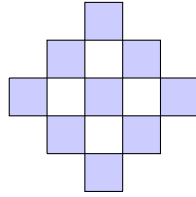
**The Even Odd Rule** The option ‘even odd rule’ is the other rule which determines which points are inside the shape. A point is considered inside the shape if a semi-infinite line originating at the point crosses an odd number of paths. Figure 5.6 depicts an example. This example also does not have self-overlapping paths.

**The Nonzero Rule** The option ‘nonzero rule’ is the default. To determine if a point,  $p$ , is inside a collection of paths, let  $c^+$  be the number of clockwise draw paths the point is in, and let  $c^-$  be the number of anticlockwise draw paths the point is in. Then  $p$  is considered inside the collection of paths if  $c^+ \neq c^-$ . Stated differently,  $p$  is inside if  $c^+ - c^- \neq 0$ , hence the name ‘non-zero rule’.

To complicate matters, closed paths may “overlap” themselves and this may result in points which are in clockwise as well as anticlockwise

**Figure 5.6**

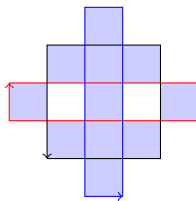
Using the ‘even odd rule’ to determine which areas are filled. There are three rectangular paths. For the ‘even odd rule’ an area is filled if it requires the crossing of an odd number of lines to get from inside the area to “infinity”.



```
\begin{tikzpicture}[fill=blue!20,scale=0.5]
\fill[even odd rule]
  (0,2) -- (0,3) -- (5,3) -- (5,2)
  (2,0) -- (3,0) -- (3,5) -- (2,5)
  (1,1) -- (4,1) -- (4,4) -- (1,4);
\draw (0,3) -- (5,3) -- (5,2) -- (0,2) -- (0,3);
\draw (3,0) -- (3,5) -- (2,5) -- (2,0) -- (3,0);
\draw (1,1) -- (4,1) -- (4,4) -- (1,4) -- (1,1);
\end{tikzpicture}
```

**Figure 5.7**

Determining the filled area with the ‘nonzero rule’. The fill involves three rectangles. One rectangle is drawn clockwise; the other rectangles are drawn anticlockwise. The red arrow corresponds to the clockwise shape; the others to the anticlockwise shapes. For the ‘nonzero rule’ a point,  $p$  is filled if  $c^+ \neq c^-$ , where  $c^+$  is the number of clockwise shapes  $p$  is in and  $c^-$  the number of anticlockwise shapes  $p$  is in. Note that  $c^+ \neq c^-$  if and only if  $c^+ - c^- \neq 0$  (hence nonzero rule).



```
\begin{tikzpicture}[fill=blue!20,scale=0.5]
\fill (0,2) -- (0,3) -- (5,3) -- (5,2)
  (2,0) -- (3,0) -- (3,5) -- (2,5)
  (1,1) -- (4,1) -- (4,4) -- (1,4);
\draw[red,->]
  (0,3) -- (5,3) -- (5,2) -- (0,2) -- (0,3);
\draw[blue,->]
  (3,0) -- (3,5) -- (2,5) -- (2,0) -- (3,0);
\draw[->]
  (1,1) -- (4,1) -- (4,4) -- (1,4) -- (1,1);
\end{tikzpicture}
```

sub-paths. To determine if a point,  $p$ , is inside the paths, let  $\ell$  be a semi-infinite line originating at  $p$ . Then  $p$  is inside the paths if the number of times  $\ell$  crosses a clockwise drawn line differs from the number of times  $\ell$  crosses an anticlockwise line. Figure 5.7 depicts an example. This example does *not* have self-overlapping paths.

## 5.9 Nodes and Node Labels

Technical pictures with lines only are rare. Usually, you want your diagrams to contain some text or math. Fortunately, `tikz` has a mechanism for adding text, math, and other typesettable material to paths. This is done using the node path operation.

```
\path ... node(<label>)[<options>]{<content>} ... ;
```

The node path extension operation places  $\langle \text{content} \rangle$  at the current position in the path using the options  $\langle \text{options} \rangle$  and associates the label  $\langle \text{label} \rangle$  with the node. The outer shape of the node is only drawn if ‘draw’ is part of  $\langle \text{options} \rangle$ . The default shape is a rectangle but other shapes are also defined. The next section explains how to control the node shape. The texts ‘ $\langle \text{label} \rangle$ ’ and ‘ $\langle \text{options} \rangle$ ’ are optional.  $\square$

```
\draw ... node(<label>)[<options>]{<content>} ... ;
```

This is similar to the previous `\path` command.  $\square$

For example, the command ‘`\draw (0,0) node {hello};`’ draws the

**Figure 5.8**  
Nodes and implicit labels.

---

```

\begin{tikzpicture}
    [every node/.style=scale=0.8]
\draw (0,0) node(hello)[scale=1.25] {hello};
\draw (hello.north) circle (2pt)
      node[anchor=south] {north};
\draw (hello.north east) circle (2pt)
      node[anchor=south west] {north east};
... % remaining commands omitted.

```

---

word ‘hello’ at the origin. Likewise, the following draws a circle and the word ‘circle’ at position (1,0).

```

\draw (0,1)          % make (0,1) current position.
      circle (2pt)    % draw circle at current position.
      node {circle}; % draw word circle at cur-
                    % rent position.

```

When a node gets a label,  $\langle \text{label} \rangle$ , then usually the additional labels  $\langle \text{label} \rangle.\text{center}$ ,  $\langle \text{label} \rangle.\text{north}$ ,  $\langle \text{label} \rangle.\text{north east}$ , ..., and  $\langle \text{label} \rangle.\text{north west}$  are also defined. The coordinates of these labels correspond to their names, so  $\langle \text{label} \rangle.\text{north}$  is to the north of the node having label  $\langle \text{label} \rangle$ . This holds for the most common node shapes. Figure 5.8 provides an example which involves all these auxiliary labels, except for  $\langle \text{label} \rangle.\text{center}$ . The option ‘anchor’ which is used in this example is explained further on. Basically, it provides a way to override the node’s default insertion point.

### 5.9.1 Predefined Nodes Shapes

The previous section demonstrated how to draw node. Nodes have a shape/style and content. The default node shape is rectangle. However, `tikz` provides many more predefined node styles such as `coordinate`, `rectangle`, `circle`, and `ellipse`. Additional node shapes may are provided by including the `tikz` library `shapes`. Some of these additional node shapes are described in the next section. The remainder of this section presents some of the basic node shapes. The shape of a node is determined by the ‘`shape =  $\langle \text{shape} \rangle$` ’ option. The following are the basic predefined node shapes:

**coordinate** This shape is for coordinates. Coordinates have no  $\langle \text{content} \rangle$ . They are not drawn but their positions are used as part of the picture.

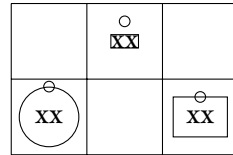
**rectangle** This shape is for rectangular nodes. The rectangle is fit around  $\langle \text{content} \rangle$ . This is the default option.

**circle** This shape is for circles. The circle is fit around  $\langle \text{content} \rangle$ .

**ellipse** This shape is for ellipses. The ellipse is fit around  $\langle \text{content} \rangle$ .



**Figure 5.9**  
Low-level node control.



```
\begin{tikzpicture}
\draw (0,0) grid (3,2);
\draw (1.5,2.5) node(a)[draw,inner sep=0pt,
                        outer sep=5pt] {xx};
\draw (3.5,1.5) node(b)[draw,inner sep=5pt,
                        outer sep=0pt] {xx};
\draw (1.5,1.5) node(c)[draw,shape=circle] {xx};
\draw (a.north) circle (2pt);
\draw (b.north) circle (2pt);
\draw (c.north) circle (2pt);
\end{tikzpicture}
```

The default height and width of a node may not always be ideal. Fortunately, there are options for low-level control. The minimum width, height, and size of a node are controlled with the options `'minimum width = <dimension>'`, `'minimum height = <dimension>'`, `'minimum size = <dimension>'`. All these options work as “expected”. Not surprisingly, there are similar options for specifying the maximum width, height, and size of a node.

There are also options to set the *inner separation* and the *outer separation* of the node. Here the *inner separation* is the extra space which is added between bounding box of the `<content>` and the node shape. For example, for a rectangular node, the inner separation determines the amount of space between the content of the node and its rectangle. Likewise, the *outer separation* is the extra space which is added to the outside of the shape of the node. Both settings affect the size of the node and the positions of the auxiliary labels ‘north’, ‘north east’, and so on. The options `'inner sep = <dimension>'` and `'outer sep = <dimension>'` set the inner and outer separation.

The options `'inner xsep = <dimension>'`, `'outer xsep = <dimension>'`, `'inner ysep = <dimension>'`, and `'outer ysep = <dimension>'` control the separations in the horizontal and vertical directions. They work “as expected”.

Figure 5.9 provides an example demonstrating some of the different node shape options and low-level control. The difference in the inner separations of the rectangular nodes manifests itself in different sizes for the rectangular shapes. Differences in the outer separations result in different distances of labels such as ‘north’. The higher the outer separation of a node, the further its ‘north’ label is away from its rectangular shape.

## 5.9.2 Node Options

This section briefly explains some of the remaining node options, which affect the drawing of nodes. The following are some of the more interesting and useful options:

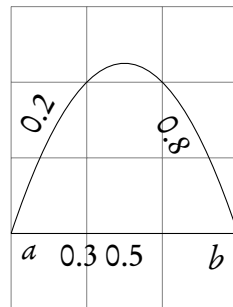
`draw`

This forces the drawing of the node shape as part of a `\path` command.

	By default the drawing of nodes is off.	□
scale=⟨factor⟩	This scales the drawing of the node content by a factor of ⟨factor⟩. This includes the font size, line widths, and so on.	□
anchor=⟨anchor⟩	This defines the <i>anchor</i> of the node. This options draws the node such that its anchor coincides with current position in the path. All node shapes define the anchor 'center', but most will also define the compass directions 'north', 'north east', 'east', ..., and 'north west'. The standard shapes also define 'base', 'base east', and 'base west'. These options are for drawing the node on its base line. The options 'mid', 'mid east', and 'mid west', which are also defined for the standard nodes, are for drawing the node on its mid anchor, which is half the height of the character 'x' above the base line. The default value for ⟨anchor⟩ is center. The anchor option is useful for relative positioning of nodes.	□
shift=⟨shift⟩	This option shifts the node in the direction ⟨shift⟩. There are also directional versions 'xshift = ⟨dimension⟩' and 'yshift = ⟨dimension⟩' for horizontal and vertical shifting.	□
above	This is equivalent to 'anchor = south'. The options 'below', 'left', 'right', 'above left', 'above right', 'below left', and 'below right' work in a similar way.	□
above=⟨shift⟩	This combines the options anchor = south and 'shift = ⟨shift⟩'. The options 'left = ⟨shift⟩', 'right = ⟨shift⟩', ..., work in a similar way.	□
rotate=⟨angle⟩	Draws the node, but rotates it ⟨angle⟩ degrees about its anchor point.	□
pos=⟨real⟩	This option is for placing nodes along a path (as opposed to at the current coordinate). This option places the node at the relative position on the path which is determined by ⟨real⟩, so if ⟨real⟩ is equal to 0.5 then the node is drawn mid-way, if it is equal to 1 then it is drawn at the end, and so on.	□
pos=sloped	This option rotates the node such that its base line is parallel to the tangent of the path at the point where the node is drawn. This option is very useful.	□
midway	This option is equivalent to using 'pos = 0.5'. Likewise, the option 'start' is equivalent to using 'pos = 0', 'very near start' is equivalent to using 'pos = 0.125', 'near start' is equivalent to using 'pos = 0.25', 'near end' is equivalent to using 'pos = 0.75', 'very near end' is equivalent to using 'pos = 0.875', and 'end' is equivalent to using 'pos = 1'.	□

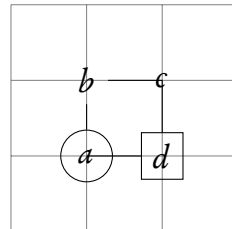
Figure 5.10 shows an example of some of these node options. Notice that several nodes can be placed with pos options for the same path segment.

**Figure 5.10**  
Node placement.



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,4);
\draw (0,1) coordinate(a)
      node[anchor=north west] {$a$}
      -- (3,1) coordinate(b)
      node[anchor=north east] {$b$}
      node[pos=0.3,anchor=north] {$0.3$}
      node[pos=0.5,anchor=north] {$0.5$}
      (a) .. controls (1,4) and (2,4) .. (b)
      node[pos=0.2,sloped,anchor=south] {$0.2$}
      node[pos=0.8,sloped,anchor=north] {$0.8$};
\end{tikzpicture}
```

**Figure 5.11**  
Drawing lines between node shapes.



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,3);
\path (1,1) node(a)[draw,shape=circle] {$a$};
\path (1,2) node(b)[shape=rectangle] {$b$};
\path (2,2) node(c)[shape=circle] {$c$};
\path (2,1) node(d)[draw,shape=rectangle] {$d$};
\draw (a) -- (b) -- (c.center) -- (d) -- (a.center);
\end{tikzpicture}
```

### 5.9.3 Connecting Nodes

The `tikz` package is well behaved. It won't cross lines unless you say so. This includes the crossing of borderlines of node shapes. For example, let's assume you've created two nodes. One of them is a circle, which is labelled  $\langle c \rangle$ , and the other is a rectangle, which is labelled  $\langle r \rangle$ . When you draw a line using the command `\draw ( $\langle c \rangle$ ) -- ( $\langle r \rangle$ );` then the resulting line segment will *not* join the centres of the two nodes. The actual line segment will be shorter because the line segment starts at the circle shape and ends at the rectangle shape. In most cases this is the desired behaviour. Should you require a line between the centres then you can always use the `'.center'` notation. Figure 5.11 provides an example.

### 5.9.4 Special Node Shapes

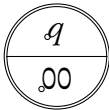
We've already seen that `tikz` has `coordinate`, `circle`, `rectangle`, and `ellipse` shape styles. Loading the `tikz library shapes` makes more shape styles available. You load this library by including it with the command `\includetikzlibrary{shapes}` in your document preamble. The following are some interesting shape styles.

`circle split`

This defines a circle with a text and a lower *node part*. The node parts of the `circle split` are separated by a horizontal line. The node part `text` is the upper and the node part `lower` is the lower part of the node

Figure 5.12

The circle split node style



```
\tikz \draw (0,0)
      node(double)[circle split,draw,double]
      {$q$ \nodepart{lower} $00$}
      (double.lower) circle (1pt)
      (double.text) circle (1pt);
```

Figure 5.13

A node with rectangle style and several parts.

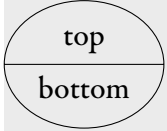
Row 1
Row 2
Row 3
Row four

```
\tikz
\node[rectangle split, rectangle split parts=4,
      every text node part/.style={align=center},
      every two node part/.style={align=left},
      every three node part/.style={align=right},
      draw, text width=2.5cm]
{ Row 1
  \nodepart{two} Row 2
  \nodepart{three} Row 3
  \nodepart{four} Row four };
```

shape circle split. The argument of the node shape determines what is in the node parts. This works as follows. You start by typesetting the default node part, which is text. Next you may switch to a different node part and typeset that node part. This may be done several times. The command `\nodepart{<part>}` switches to the node part `<part>`. After switching to the node part `<part>` you provide commands that typeset `<part>`. For example, `\node[shape=circle split]{top \nodepart{lower} bottom}` typesets a circle split whose text part has ‘top’ in it and whose lower part has ‘bottom’ in it. The node shape `circle split` inherits all labels from the node shape `circle`. It also gets a label for the lower part. The following is an example. The node option `double` in this example results in a circle with a double line. Figure 5.12 provides an example. □

ellipse split

This is the ellipse version of circle split. The following is an example.



```
\tikz
\draw (0,0)
      node[ellipse split,draw]
      {top
       \nodepart{lower}
       bottom};
```

□

rectangle split

This is the rectangle version of circle split. However, the rectangle version is more versatile. The rectangle can split horizontally or vertically into up to 20 parts. There are quite a number options for this shape. The example in Figure 5.13 draws a rectangle with four parts. The example won’t work if the text width isn’t set explicitly. The reader is referred to the `tikz` manual [Tantau, 2010] for further information. □

## 5.10 Coordinate Systems

The key to effective, efficient, and maintainable picture creation is the ability to specify coordinates. Coordinates may be specified in different ways each coming with its own specific *coordinate system*. Within a coordinate system you specify coordinates using *explicit* or *implicit* notation.

**Explicit** Explicit coordinate specifications are verbose. To specify a coordinate, you write ‘ $(\langle \text{system} \rangle \text{ cs} : \langle \text{coord} \rangle)$ ’, where  $\langle \text{system} \rangle$  is the name of the coordinate system and where  $\langle \text{coord} \rangle$  is a coordinate whose syntax depends on  $\langle \text{system} \rangle$ . For example, to specify the point having  $x$ -coordinate  $\langle x \rangle$  and  $y$ -coordinate  $\langle y \rangle$  in the canvas coordinate system you write ‘ $(\text{canvas cs} : x = \langle x \rangle, y = \langle y \rangle)$ ’.

**Implicit** Implicit coordinates specifications are usually more terse than explicit coordinate specifications. Here, you specify coordinates using some coordinate system-specific notation inside parentheses. Most examples so far have used the implicit notation for the canvas coordinate system.

The remainder of this section studies some of the more useful coordinates systems. The notation for explicit coordinate specification being too verbose, we shall focus on using implicit notation.

**Canvas Coordinate System** The most widely used coordinate system is the ‘canvas’ coordinate system. It defines coordinates in terms of a horizontal and a vertical offset relative to the origin. The implicit notation ‘ $(\langle x \rangle, \langle y \rangle)$ ’ is the point with  $x$ -coordinate  $\langle x \rangle$  and  $y$ -coordinate  $\langle y \rangle$ .

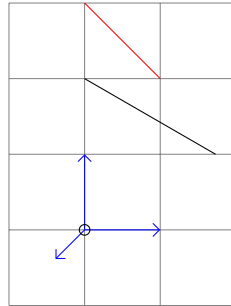
**XYZ Coordinate System** The ‘xyz’ coordinate system defines coordinates in terms of a linear combination of an  $x$ -, a  $y$ -, and a  $z$ -vector. By default, the  $x$ -vector points 1 cm to the right, the  $y$ -vector points 1 cm up, and the  $z$ -vector points to  $(-\sqrt{2}/2, -\sqrt{2}/2)$ . However, these default settings can be changed. The implicit notation ‘ $(\langle x \rangle, \langle y \rangle, \langle z \rangle)$ ’ is used to define the point which is located at  $\langle x \rangle$  times the  $x$ -vector plus  $\langle y \rangle$  times the  $y$ -vector plus  $\langle z \rangle$  times the  $z$ -vector.

**Polar Coordinate System** The ‘canvas polar’ coordinate system defines coordinates in terms of an angle and a radius. The implicit notation ‘ $(\alpha : r)$ ’ corresponds to the point  $r \times (\cos \alpha, \sin \alpha)$ . Angles in this coordinate system, as all angles in `tikz`, should be supplied in degrees.

**Node Coordinate System** The ‘node’ coordinate system defines coordinates in terms of a label of a node or coordinate. The implicit notation ‘ $(\langle \text{label} \rangle)$ ’ is the position of the node or coordinate which was given the label  $\langle \text{label} \rangle$ .

**Figure 5.14**

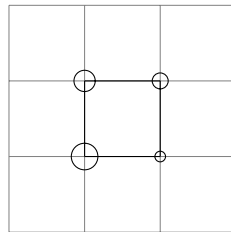
Using four coordinate systems.



```
\begin{tikzpicture}[>=angle 90]
\draw[help lines] (-1,-1) grid (2,3);
\draw[red] (canvas cs:x=1cm,y=2cm) -- (0,3);
\draw[blue,->] (0,0) -- (xyz cs:x=1,y=0,z=0);
\draw[blue,->] (0,0) -- (0,1,0);
\draw[blue,->] (0,0) -- (0,0,1);
\draw (canvas polar cs:radius=2cm,angle=30)
-- (90:2);
\path (0,0) coordinate (origin);
\draw (origin) node circle (2pt);
\end{tikzpicture}
```

**Figure 5.15**

Computing the intersection of perpendicular lines.



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid +(3,3);
\path (1,1) coordinate (l1);
\path (2,2) coordinate (ur);
\draw (l1) -- (l1 -| ur) circle (2pt);
\draw (l1 -| ur) -- (ur) circle (3pt);
\draw (ur) -- (ur -| l1) circle (4pt);
\draw (ur -| l1) -- (l1) circle (5pt);
\end{tikzpicture}
```

Figure 5.14 demonstrates the previous four coordinate systems in action. The optional argument of the `tikzpicture` sets the arrow head style to the predefined style ‘`angle 90`’.

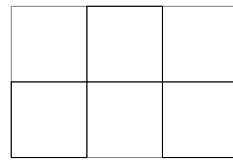
**Perpendicular Coordinate System** The ‘perpendicular’ coordinate system is a dedicated system for computing intersections of horizontal and vertical lines. With this coordinate system’s implicit syntax you write ‘ $(\langle \text{pos}_1 \rangle \mid - \langle \text{pos}_2 \rangle)$ ’ for the coordinate at the intersection of the infinite vertical line through  $\langle \text{pos}_1 \rangle$  and the infinite horizontal line through  $\langle \text{pos}_2 \rangle$ . Likewise, ‘ $(\langle \text{pos}_1 \rangle - \mid \langle \text{pos}_2 \rangle)$ ’ results in the intersection of the infinite horizontal line through  $\langle \text{pos}_1 \rangle$  and the infinite vertical line through  $\langle \text{pos}_2 \rangle$ . The notation for this coordinate system is quite suggestive as ‘ $\mid$ ’ suggests the vertical aspect of the line and ‘ $-$ ’ suggests the horizontal aspect of the other line. The order of the lines is then given by the order inside the operators ‘ $\mid -$ ’ and ‘ $- \mid$ ’. Inside the parentheses you are not supposed to use parentheses for coordinates and labels, so you write ‘ $(0,1 \mid - 1,2)$ ’, ‘ $(\text{label} \mid - 1,2)$ ’, and so on. Figure 5.15 demonstrates how to use the perpendicular coordinate system.

## 5.11 Coordinate Calculations

Specifying diagrams in terms of absolute coordinates is cumbersome and prone to errors. What is worse, diagrams defined in terms of absolute coordinates are difficult to maintain. For example, changing the position of an  $n$ -agon which is defined in terms of absolute coordinates requires

**Figure 5.16**

Absolute, relative, and incremental coordinates.




---

```

\begin{tikzpicture}
\draw[help lines] (0,0) grid +(3,2);
\draw (0,0) -- (+1,0) --
      (1,1) -- (+0,1) -- cycle;
\draw (1,1) -- +(1,0) --
      +(1,1) -- +(0,1) -- cycle;
\draw (2,0) -- ++(1,0) --
      ++(0,1) -- ++(-1,0) -- cycle;
\end{tikzpicture}

```

---

changing  $n$  coordinates. Fortunately, `tikz` lets you compute coordinates from other coordinates. Used intelligently, this helps reducing the maintenance costs of your diagrams.

There are two kinds of coordinate computations. The first kind involves *relative* and *incremental* coordinates. These computations depend on the current coordinate in a path. They are explained in Section 5.11.1. The second kind of computations are more general. They can be used to compute coordinates from one or several given coordinates, relative or absolute distances, rotation angles, and projections. These computations are explained in Section 5.11.2.

### 5.11.1 Relative and Incremental Coordinates

*Relative* and *incremental* coordinates are coordinates which are computed relative to the current coordinate in a path. The first doesn't change the current coordinate whereas the second does change it.

**Relative coordinate** A relative coordinate constructs a new coordinate at an offset from the current coordinate without changing the current coordinate. The notation '`+<offset>`' specifies the relative coordinate which is located at offset `<offset>` from the current coordinate.

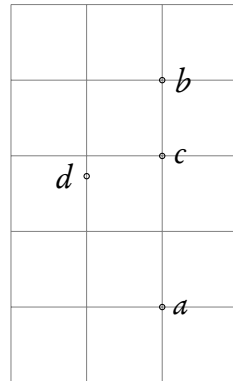
**Incremental coordinate** An incremental also coordinate constructs a new coordinate at an offset from the current coordinate. This time, however, the new coordinate becomes the current coordinate. You use the implicit notation '`++<offset>`' for incremental coordinates.

Figure 5.16 provides an example that draws three squares. The first square is drawn using absolute coordinates, the second with relative coordinates, and the last with incremental coordinates. Clearly, the drawing with relative and incremental coordinates should be preferred as it improves the maintenance of the picture. For example, moving the first square requires changing four coordinates, whereas moving the second or third square requires changing only the start coordinate. Using a relative coordinate also improves the maintainability of the grid.

### 5.11.2 Complex Coordinate Calculations

**Figure 5.17**

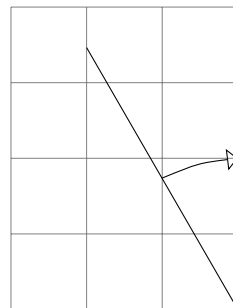
Coordinate computations with partway modifiers.



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid +(3,5);
\draw (2.0,1.0) circle (1pt)
      coordinate(a)
      node[anchor=west] {$a$}
      (2.0,4.0) circle (1pt)
      coordinate(b)
      node[anchor=west] {$b$}
      ($ (a)!0.666!(b)$ ) circle (1pt)
      node[anchor=west] {$c$}
      ($ (a)!0.666!30:(b)$ ) circle (1pt)
      node[anchor=east] {$d$};
\end{tikzpicture}
```

**Figure 5.18**

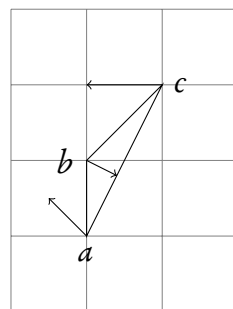
Coordinate computations with partway and distance modifiers.



```
\begin{tikzpicture}
\draw[help lines] (-3,0) grid +(3,4);
\draw (0,0) --
      ($ (0,0)! 1! 30:(0,4)$ ) coordinate(a)
      ($ (0,0)!2cm! (a)$ ) coordinate(b)
      ($ (0,0)!2cm!-15:(a)$ ) coordinate(c)
      ($ (0,0)!2cm!-30:(a)$ ) coordinate(d);
\draw[-open triangle 90]
      (b) .. controls (c) .. (d);
\end{tikzpicture}
```

**Figure 5.19**

Coordinate computations with projection modifiers.



```
\begin{tikzpicture}[>=open triangle 90]
\draw[help lines] (0,0) grid +(3,4);
\draw (1,1) coordinate(a) node[anchor=north] {$a$}
      -- (1,2) coordinate(b) node[anchor=east] {$b$}
      -- (2,3) coordinate(c) node[anchor=west] {$c$}
      -- cycle;
\draw[->] (b) -- ($ (a)!(b)!(c)$ );
\draw[->] (c) -- ($ (b)!(c)!(a)$ );
\draw[->] (a) -- ($ (c)!(a)!(b)$ );
\end{tikzpicture}
```

Finally, `tikz` offers complex coordinate calculations. However, these calculations are only available if the `tikz` library `calc` is loaded in the preamble: `\usetikzlibrary{calc}`.

([ $\langle$ options $\rangle$ ]) $\langle$ coordinate computation $\rangle$ )

This is the general syntax. The  $\langle$ coordinate computation $\rangle$  should:

1. Start with ' $\langle$ factor $\rangle * \langle$ coordinate $\rangle \langle$ modifiers $\rangle$ '. Here  $\langle$ modifiers $\rangle$  is a sequence of one or more  $\langle$ modifier $\rangle$ s and ' $\langle$ factor $\rangle *$ ' is an optional multiplication factor which defaults to 1. Both are described further on.



2. Continue with one or more expressions of the form: ‘ $\langle \text{sign option} \rangle \langle \text{factor} \rangle * \langle \text{coordinate} \rangle \langle \text{modifiers} \rangle$ ’, where  $\langle \text{sign option} \rangle$  is an optional ‘+’ or ‘-’.

□

 $\langle \text{factor} \rangle$ 

Each  $\langle \text{factor} \rangle$  is an optional numeric expression which is parsed by the `\pgfmathparse` command. Examples of valid  $\langle \text{factor} \rangle$ s are ‘1.2’, ‘ $\{3 * 4\}$ ’, ‘ $\{3 * \sin(60)\}$ ’, ‘ $\{3 + (2 * 4)\}$ ’, and so on. Inside the braces it is safe to use parentheses, except for the top level. The reason why parentheses do not work at the top level is that  $\langle \text{factor} \rangle$ s are optional and that the opening parentheses are reserved for the start of a coordinate. Therefore, compound expressions at the top level are best put inside braces as this makes parsing easier at the top level.

□

 $\langle \text{modifier} \rangle$ 

A  $\langle \text{modifier} \rangle$  is a postfix operator which acts on the coordinate preceding it. There are three different kinds:  $\langle \text{pmod} \rangle$ ,  $\langle \text{dmod} \rangle$ , and  $\langle \text{prmod} \rangle$ . Each of them is of the form ‘ $\langle \text{stuff} \rangle$ ’ and it is used after a coordinate. To explain the modifiers we shall write  $\langle \text{partway modifier} \rangle$  for  $\langle \text{coordinate} \rangle ! \langle \text{pmod} \rangle$ , shall write  $\langle \text{distance modifier} \rangle$  for  $\langle \text{coordinate} \rangle ! \langle \text{dmod} \rangle$ , and shall write  $\langle \text{projection modifier} \rangle$  for  $\langle \text{coordinate} \rangle ! \langle \text{prmod} \rangle$ . □

 $\langle \text{partway modifier} \rangle$ 

There are two different forms of  $\langle \text{partway modifier} \rangle$ s. The first form is ‘ $\langle \text{coordinate}_1 \rangle ! \langle \text{factor} \rangle ! \langle \text{coordinate}_2 \rangle$ ’. The resulting coordinate is given by

$$\langle \text{coordinate}_1 \rangle + \langle \text{factor} \rangle \times (\langle \text{coordinate}_2 \rangle - \langle \text{coordinate}_1 \rangle).$$

In words this is the coordinate which is at  $\langle \text{factor} \rangle \times 100\%$  distance along the line between  $\langle \text{coordinate}_1 \rangle$  and  $\langle \text{coordinate}_2 \rangle$ .

A  $\langle \text{partway modifier} \rangle$  may also have the complex form ‘ $\langle \text{coordinate}_1 \rangle ! \langle \text{factor} \rangle ! \langle \text{angle} \rangle : \langle \text{coordinate}_2 \rangle$ ’. The result of this complex form is given by first computing  $\langle \text{coordinate}_1 \rangle ! \langle \text{factor} \rangle ! \langle \text{coordinate}_2 \rangle$  and rotating the resulting coordinate about  $\langle \text{coordinate}_1 \rangle$  over  $\langle \text{angle} \rangle$  degrees. Figure 5.17 presents an example of coordinate computations involving partway modifiers. □

 $\langle \text{distance modifier} \rangle$ 

The next modifier is the  $\langle \text{distance modifier} \rangle$ . This modifier has the form ‘ $\langle \text{coordinate}_1 \rangle ! \langle \text{distance} \rangle : \langle \text{angle} \rangle ! \langle \text{coordinate}_2 \rangle$ ’, where ‘ $: \langle \text{angle} \rangle$ ’ is optional.

The simpler form ‘ $\langle \text{coordinate}_1 \rangle ! \langle \text{distance} \rangle ! \langle \text{coordinate}_2 \rangle$ ’ results in the coordinate which is at distance  $\langle \text{distance} \rangle$  from  $\langle \text{coordinate}_1 \rangle$  in the direction from  $\langle \text{coordinate}_1 \rangle$  to  $\langle \text{coordinate}_2 \rangle$ . For example, if the two coordinates are at distance 2 cm apart then setting  $\langle \text{distance} \rangle$  to 1 cm gives you the point halfway between the two coordinates.

The more complex form of the distance modifier is similar and works in a similar way as the partway modifier. This time you write ‘ $\langle \text{coordinate}_1 \rangle ! \langle \text{distance} \rangle : \langle \text{angle} \rangle ! \langle \text{coordinate}_2 \rangle$ ’. The result is obtained by first computing  $\langle \text{coordinate}_1 \rangle ! \langle \text{distance} \rangle ! \langle \text{coordinate}_2 \rangle$

and rotating the result about  $\langle \text{coordinate}_1 \rangle$  over  $\langle \text{angle} \rangle$  degrees in counter-clockwise direction. Figure 5.18 presents an example of coordinate computations involving distance modifiers.  $\square$

$\langle \text{projection modifier} \rangle$

The final  $\langle \text{modifier} \rangle$  is  $\langle \text{projection modifier} \rangle$ . This modifier is of the form ‘ $\langle \text{coordinate}_1 \rangle! \langle \text{coordinate}_2 \rangle! \langle \text{coordinate}_3 \rangle$ ’<sup>1</sup> and it results in the projection of  $\langle \text{coordinate}_2 \rangle$  on the infinite line through  $\langle \text{coordinate}_1 \rangle$  and  $\langle \text{coordinate}_3 \rangle$ . Figure 5.19 presents an example of coordinate computations with projection modifiers. (For some reason my tikz version doesn’t like extra space around ‘!’ inside a  $\langle \text{projection modifier} \rangle$ . It is not clear whether this is a feature.)  $\square$

## 5.12 Options

Many tikz commands and environments depend on options. Usually these options are specified using ‘ $\langle \text{key} \rangle = \langle \text{value} \rangle$ ’ combinations. Some combinations have shorthand notations. For example, ‘ $\langle \text{colour} \rangle$ ’ is a shorthand notation for ‘ $\langle \text{color} \rangle = \langle \text{colour} \rangle$ ’. Options are best defined by passing their  $\langle \text{key} \rangle = \langle \text{value} \rangle$  combinations as part of the optional argument. However, there is another mechanism.

$\backslash \text{tikzset} \{ \langle \text{options} \rangle \}$

Sets the options in  $\langle \text{options} \rangle$ . The options are set using the pgfkeys package. This package is quite powerful but explaining it goes beyond the scope of an introduction like this chapter. Roughly speaking, processing the keys works “as expected” for “normal” usage.  $\square$

## 5.13 Styles

One of the great features of tikz is *styles*. Defining a style for your graphics has several advantages.

**Control** You can use styles to control the appearance. For example, by carefully designing a style for drawing auxiliary lines, you can draw them in a style which makes them appear less prominently in the picture. Other styles may be used to draw lines which should stand out and draw attention.

**Consistency** Drawing and colouring sub-parts with a carefully chosen style guarantees a consistent appearance of your diagrams. For example, if you consistently draw help lines in a dedicated, easily recognisable style then it makes it easier to recognise them.

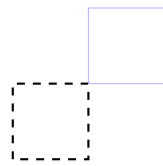
**Reusability** Styles which are defined once can be reused several times.

**Simplicity** Changing the appearance of a graphical element with styles with well-understood interfaces is much easier and leads to fewer errors.

---

<sup>1</sup>The manual on Page 119 also mentions an angle but it is not explained how to use it....

**Figure 5.20**  
Predefining options with the  
`\tikzset` command.



```
\tikzset{thick dashed/.style={thick,dashed}}
\begin{tikzpicture}
    [{help lines/.style={ultra thin,blue!30}}]
\draw[thick dashed] (0,0) rectangle (1,1);
\draw[help lines] (1,1) rectangle (2,2);
\end{tikzpicture}
```

**Refinement** You can stepwise refine the way certain graphics are drawn. This lets you postpone certain design decisions while still letting you draw your diagrams in terms of the style. By refining the style at a later stage, you can fine-tune the drawing of all the relevant sub-graphics.

**Maintainability** This advantage is related to the previous item. Unforeseen changes in global requirements can be implemented by making a few local changes.

Styles affect options. For example, the predefined ‘help lines’ style sets `draw` to ‘black!50’ and sets ‘line with’ to ‘very thin’. You can set a style at a global or at a local level. The following command defines a style at a global level.

```
\tikzset{<style name>/.style={<list>}}
```

This defines a new style `<style name>` and gives it the value `<list>`, where `<list>` is a list defining the style. In its basic usage `<list>` is a list of ground options, but it is also possible to define styles which take arguments. The following example defines a style `Cork` which sets `draw` to red and uses a thick line.



```
\tikzset{Cork/.style={draw=red,thick}}
\tikz \draw[Cork]
    (0,0) rectangle (1,1);
```

Defining a style at a local level is done by passing a ‘`<style name>`’ = `<list>`’ as an option.

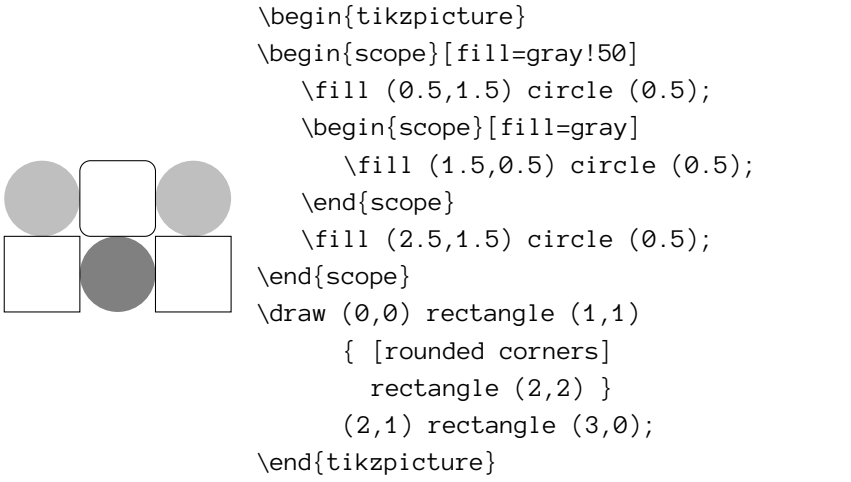
```
<style name>/.style={<list>}
```

This defines `<style name>` in the scope of the environment or command which takes the option. This mechanism may also be used to temporarily override the existing definition of `<style name>`. Figure 5.20 provides an example. □

## 5.14 Scopes

Scopes in `tikzpicture` environments serve a similar purpose as blocks in a programming language and groups in `TEX`. They allow you to temporarily change certain settings upon entering the scope and restore the previous settings when leaving the scope. In addition `tikz` scopes let you execute code at the start and end of a scope. Scopes in `tikz` are implemented as an environment called ‘scope’. Scopes depend on the following style.

**Figure 5.21**  
Using scopes



every scope

This style is installed at the start of every scope. The style is empty initially. Using the mechanisms which are explained in the previous section you can either set the value of this style using `every scope/.style={}` or set the style with the options of a `tikzpicture` environment.

```
\begin{tikzpicture}[every scope/.style={\list}]
...
\end{tikzpicture}
```

LaTeX Usage

The following options allow you to execute code at the start and end of the scope.

execute at begin scope=`<code>` This option results in executing `<code>` at the start of the scope. ☐

execute at end scope=`<code>` This option results in executing `<code>` at the end of the scope. ☐

The `tikz` library `scopes` defines a shorthand notation for scopes. It lets you write `{ [ <options> ] <stuff> }` for `\begin{scope} [ <options> ] <stuff> \end{scope}`. Interestingly you can also have scopes *inside* paths. However, options of a local scope in a path do not affect path options such as line thickness, colour and so on which apply to the whole path. Figure 5.21 depicts an example.

### 5.15 The `\foreach` Command

As if `tikz` productivity isn't enough, its `pgf` for library provides a very flexible `foreach` command.

`\foreach <macros> in {<list>} {<statements>}`  
Here `<macros>` is a forward slash-delimited list of macros and `<list>` is a comma-delimited list consisting of lists of forward slash-delimited values. For each list of values in `<list>`, the `\foreach` command binds the *i*-th value of the list to the *i*-th macro in `<macros>` and then carries out `<statements>`. Figure 5.22 depicts an example. Notice that the previous

**Figure 5.22**  
The `\foreach` command.

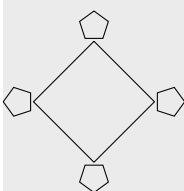
		<code>\tikz</code>
4	3	<code>\foreach \pos/\text in</code>
		<code>{\{0,0\}/1,\{1,0\}/2,\{1,1\}/3,\{0,1\}/4}</code>
1	2	<code>{</code>
		<code>\draw (\pos) node {\text};</code>
		<code>}</code>

**Table 5.3**  
Shorthand notation for the `\foreach` command. The notation in the upper part of the table involves ranges which depend on an initial value and a next value which determines the increment. The shorthand notation in the middle part depends only on the initial value and the final value in the range. Here the increment is 1 if the final value is greater than the initial value. Otherwise the increment is  $-1$ . The lower part of the table demonstrates the `\foreach` command also allows pattern-matching.

Command	Yields
<code>\foreach \x in {1,2,...,6} {\x,}</code>	1, 2, 3, 4, 5, 6,
<code>\foreach \x in {1,3,...,10} {\x,}</code>	1, 3, 5, 7, 9,
<code>\foreach \x in {1,3,...,11} {\x,}</code>	1, 3, 5, 7, 9, 11,
<code>\foreach \x in {0,0.1,...,0.3} {\x,}</code>	0, 0.1, 0.20001, 0.30002,
<code>\foreach \x in {a,b,...,d,9,8,...,6} {\x,}</code>	a, b, c, d, 9, 8, 7, 6,
<code>\foreach \x in {7,5,...,0} {\x,}</code>	7, 5, 3, 1,
<code>\foreach \x in {Z,X,...,M} {\x,}</code>	Z, X, V, T, R, P, N,
<code>\foreach \x in {1,...,5} {\x,}</code>	1, 2, 3, 4, 5,
<code>\foreach \x in {5,...,1} {\x,}</code>	5, 4, 3, 2, 1,
<code>\foreach \x in {a,...,e} {\x,}</code>	a, b, c, d, e,
<code>\foreach \x in {2^1,2^... ,2^6} {\\$ \x\$,}</code>	$2^1, 2^2, 2^3, 2^4, 2^5, 2^6$
<code>\foreach \x in {0\pi,0.5\pi,...\pi,2\pi} {\\$ \x\$,}</code>	$0\pi, 0.5\pi, 1.5\pi, 2.0\pi,$
<code>\foreach \x in {A_1,..._1,D_1} {\\$ \x\$,}</code>	$A_1, B_1, C_1, D_1,$

example demonstrates that grouping may be used to construct values in `\list` with commas in them. In general this is a useful technique. However, since coordinates are very common, there is no need to turn coordinates into a group.

It is also possible to use `\foreach` inside the `\path` command. The following is an example, which also demonstrates that `tikz` also supports a limited form of arithmetic.

	<code>\tikz \draw (0,-0.8)</code>
	<code>\foreach \angle in {0,90,180,270} {</code>
	<code>-- (\angle:0.8)</code>
	<code>(\angle:1.0) + (\angle:0.2)</code>
	<code>\foreach \fraction in {1,2,3,4,5} {</code>
	<code>-- +(\angle+\fraction*72:0.2)</code>
	<code>} -- cycle (\angle:0.8)</code>
	<code>};</code>
	□

If there is only one macro in `\macros` then shorthand notations are allowed in `\list` which may be used for “regular” lists of values. Some examples of these shorthand notations are listed in Table 5.3. The table is based on the `tikz` documentation.

## 5.16 The let Operation

The `let` operation binds expressions to “variables” inside a path. The following is the general syntax.


```
\path ... let <assignments> in ...;
```

Here `<assignments>` is a comma-delimited list of assignments. Each assignment is of the form ‘`<register> = <expression>`’. To carry out the assignment, `<expression>` is evaluated and then assigned to `<register>`, which is some variable which is local to `tikz`. After the assignments, the values of the variables may be got using the macros `\n`, `\p`, `\x`, and `\y`. However, this is only possible in the scope of the assignment lasting from ‘`in`’ to the semicolon at the end of the path operation. The assignment mechanism respects the `tikz` scoping rules. □

There are two kinds of `<register>`s in assignments of the `let` operation. Both are written as macro calls. The first kind are *number registers*, which are written as `\n{<name>}`. Here `<name>` is just a convenient label, which may be almost any combination of characters, digits, space characters, and other symbols, except for special characters and the dot. As the name suggests, number registers store numeric values. The second kind of register is the *point register*. Point registers start with `\p{<name>}`. They store coordinate values. The following explains both `<register>`s in more detail.

```
\n{<name>} = <expression>
```

This is for assigning a numeric value to the number register `<name>`. The command `\n{<name>}` returns the current value of the number register `<name>`. The following shows how to use number registers.




```
\tikz
\draw let \n0 = 0, \n1 = 1 in
      let \n{the sum} = \n0 + \n1 in
      (0,0) -- (\n1,\n0) --
      (\n1,\n{the sum});
```

□

```
\p{<name>} = <expression>
```

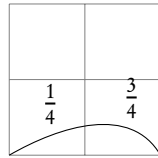
This is for assigning points to the point register `<name>`. The command `\p{<name>}` returns the current (point) value of the point register `<name>`. The *x*- and *y*-coordinates of the point register may be got with the commands `\x{<name>}` and `\y{<name>}`. The following is an example which assumes that the `tikz` library `calc` has been loaded. Note that this example can also be written with a single `let` operation.



```
\tikz \draw
let \p{ll} = (0,0),
  \p{ur} = (1,1) in
let \p{ul} = (\x{ll},\y{ur}),
  \p{lr} = ($\p{ll}!1!90:\p{ul}$) in
(\p{ll}) -- (\p{lr}) -- (\p{ur}) --
(\p{ul}) -- cycle;
```

□

**Figure 5.23**  
Simple to path example.



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (2,2);
\draw[out=30,in=120]
  (0,0) to node[pos=0.25,above] {\frac{1}{4}}
        node[pos=0.75,above] {\frac{3}{4}}
        (2,0);
\end{tikzpicture}
```

## 5.17 The To Path Operation

This section describes the ‘to’ path operation which lets you connect two nodes in a given style. For example, writing ‘\path (0,0) to (1,0);’ give you the same as ‘\path (0,0) -- (1,0);’ but ‘\path (0,0) to[out=45,in=135] (1,0);’ connects the points with an arc which leaves the point (0,0) at 45° and enters (1,0) at 135°.

It is also possible to define styles for to operations. This lets you draw complex paths with a single operation. The general syntax of the to path operation is as follows.

```
\path ... to[⟨options⟩] ⟨nodes⟩ (⟨coordinate⟩) ...;
```

The ⟨nodes⟩ are optional nodes which are placed on the path. Figure 5.23 presents an example. □

The following style is defined for to paths.

every to

It is installed at the beginning of every to path. The following is an example.



```
\tikzset{every to/.style={out=90,in=180}}

\begin{tikzpicture}
\draw[help lines] (0,0) grid (2,2);
\draw (0,0) to (2,2)
      (0,0) to (1,1);
\end{tikzpicture}
```

Options affect the style of to paths. The following is arguably the more important style.

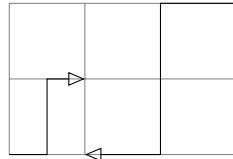
to path=⟨path⟩

With this option, the following path is inserted: ‘{[every to,⟨options⟩] ⟨path⟩}’. Here ⟨options⟩ are the options passed to the to path. Inside ⟨path⟩ you can use the commands \tikztostart, \tikztotarget, and \tikztonodes. The value of \tikztostart is the start node of and that of \tikztotarget the end node. The value of \tikztonodes is given by ⟨nodes⟩, i.e. the nodes of the to path. It should be noted that the values returned by \tikztostart and \tikztotarget do *not* include parentheses. □

Figure 5.24 demonstrates how to define a ‘to path’ style called ‘hvh’ (for horizontal, vertical, horizontal), which may be used to connect two points using three line segments. The first line segment is horizontal, the

**Figure 5.24**

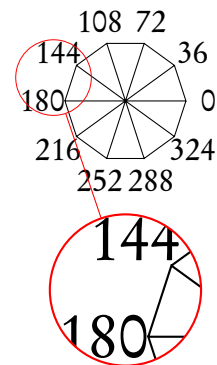
A user-defined ‘to path’ style. The `\tikzset` on Line 1 in the input defines a new ‘to path’ style called `hvh`. Lines 2–5 define the actions of the style. The commands `\tikztostart`, `\tikztotarget`, and `\tikztonodes` are determined by the `\path` command. The command `\tikztostart` is the start of the path, `\tikztotarget` is the destination of the path, and `\tikztonodes` is the (optional) nodes at the end of the path.



```
\tikzset{hvh/.style={to path={
let \p{mid}=(\tikztostart)!0.5!(\tikztotarget)$
in -- (\tikztostart -| \p{mid})
-- (\p{mid} |- \tikztotarget)
-- (\tikztotarget) \tikztonodes}}
\begin{tikzpicture}
\draw[help lines] (0,0) grid +(3,2);
\draw[-open triangle 45] (0,0) to[hvh] (1,1);
\draw[-open triangle 45] (3,2) to[hvh] (1,0);
\end{tikzpicture}
```

**Figure 5.25**

Using the `spy` library.



```
\begin{tikzpicture}
[spy using outlines={circle,
magnification=2,
size=2cm,
connect spies}]
\draw (-36:0.8)
\foreach \angle in {0,36,...,359} {
-- (\angle:0.8)
(\angle:1.1) node {\angle}
(0,0) -- (\angle:0.8)
};
\spy[red] on (162:1.0) in node[right] at (0,-2.5);
\end{tikzpicture}
```

second is vertical, and the third is horizontal.

## 5.18 The `spy` Library

The `spy` library lets you magnify parts of diagrams. These magnifications are technically known as *canvas transformations*, which means they affect everything, including line widths, font size, and so on.

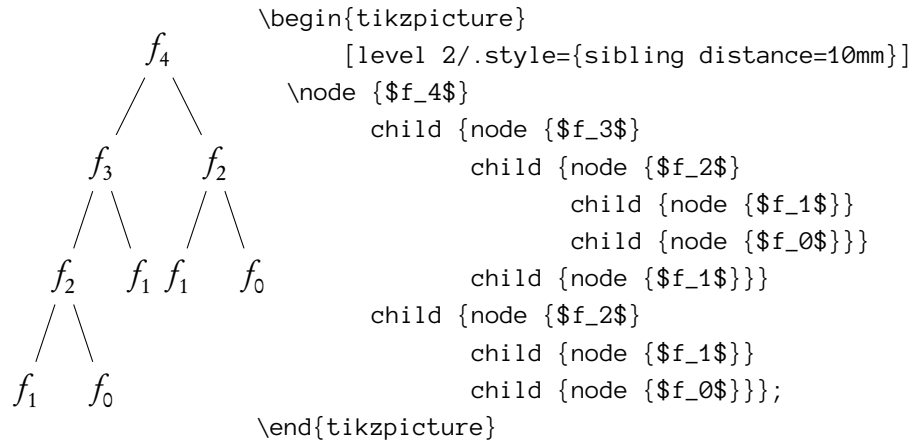
To use the feature you need to add the option ‘`spy scope`’ to the picture or scope you wish to spy upon. Some options implicitly add this option. I’ve noticed problems with the `spy` feature and `xelatex`. Fortunately it work flawlessly with `pdflatex`. Figure 5.25 provides an example. The `spy` library has quite a number of options. If you like to spy on your `tikzpictures` then you may find more details in the manual [Tantau, 2010].

## 5.19 Trees

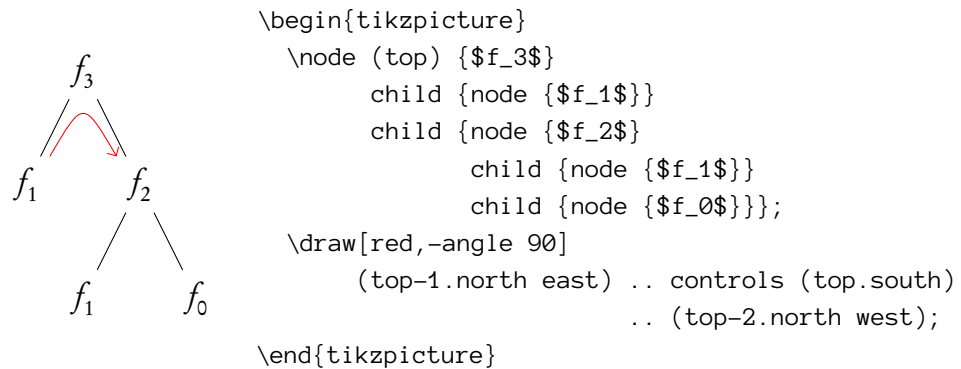
Knowing how to define node labels and knowing how to draw nodes and basic shapes, we are ready to draw some more interesting objects.



**Figure 5.26**  
Drawing a tree.



**Figure 5.27**  
Using implicit node labels in trees. To draw the arrow, the label of the root node is used to construct the labels of its first and second child.



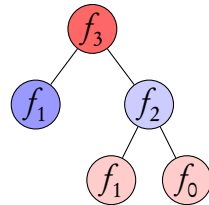
We shall start with a class of objects which should be of interest to the majority of computer scientists: trees.

Trees expose a common theme in `tikz` objects: hierarchical structures. A tree is defined by defining its root and the children of each node in the tree. Each child is a node or a node with children. By default, the children of each parent are drawn from left to right in order of appearance. Unfortunately, drawing trees with `tikz` isn't perfect. The 'sibling distance =  $\langle \text{dimension} \rangle$ ' option lets you control the sibling distance. You can control these distances globally or for a fixed level. For example, 'level 2/.style = {sibling distance = 1cm}' sets the distance for the grandchildren of the root — they are at level 2 — to 1 cm. Figure 5.26 demonstrates how to draw a tree.

Inside trees you can use labels as usual. What is more, `tikz` implicitly labels the nodes in the tree and lets you use these labels. The  $i$ th child of a parent with label  $\langle \text{parent} \rangle$  is labelled  $\langle \text{parent} \rangle-i$ . This process is continued recursively, so the  $j$ th child of the  $i$ th child of the parent node is labelled  $\langle \text{parent} \rangle-i-j$ . Figure 5.27 demonstrates the mechanism.

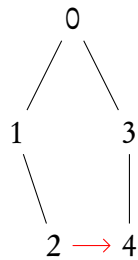
Changing the node style is a piece of cake. Figure 5.28 provides an example which sets the styles of the second and third level to different defaults. The option 'level distance =  $\langle \text{dimension} \rangle$ ' sets the distance between the levels in the tree.

**Figure 5.28**  
Controlling the node style.



```
\begin{tikzpicture}
  [level distance=10mm%
  ,every node/.style={fill=red!60%
    circle,%
    draw=black,%
    inner sep=1pt}%
  ,level 1/.style={sibling distance=15mm},%
  ,level 2/.style={sibling distance=10mm,%
    nodes={fill=red!20}}]
  \node (top) {$f_3$}
    child {node[fill=blue!40] {$f_1$}}
    child {node[fill=blue!20] {$f_2$}}
      child {node {$f_1$}}
      child {node {$f_0$}};
\end{tikzpicture}
```

**Figure 5.29**  
A tree with a ‘missing’ node.  
The node of the first child of  
the root’s first child is left out  
using the node option `missing`.



```
\begin{tikzpicture}
  [level 2/.style={sibling distance=10mm}]
  \node (top) {$0$}
    child {node {$1$}}
      child[missing]
      child {node {$2$}}
    child {node {$3$}}
      child {node {$4$}};
  \draw[red,-angle 90]
    (top-1-2.east) -- (top-2-1.west);
\end{tikzpicture}
```

As already noted, the rules for automatic node placement are not always ideal. For example, sometimes you may wish to have the single child of a given parent drawn to the left or to the right of the parent. The `child` option `missing` allows you to specify a node which takes up space but which is not drawn. By putting such a node to the left of its sibling, the position of the sibling is forced to the right. You may use this mechanism to force node placement. Omitting a node makes its label inaccessible. Figure 5.29 provides an example.

## 5.20 Logical Circuits

A *logical circuit* is a circuit whose building blocks are logic gates such as and-gates, or-gates, xor-gates (exclusive or), not-gates, and so on. Needless to say that `tikz` lets you draw logical circuits with ease. The style of the symbols of the gates depends on libraries. Possible libraries are `circuits.logic.IEC`, `circuits.logic.CDH`, and `circuits.logic.US`. You may load these libraries with the command `\usetikzlibrary` command. Table 5.4 demonstrates the node shapes you get with the different

**Table 5.4**  
Node shapes provided by  
logic gate shape libraries.

Node Shape	Appearance			Node Shape	Appearance		
	IEC	CDH	US		IEC	CDH	US
and gate				nand gate			
or gate				nor gate			
xor gate				xnor gate			
not gate				buffer gate			

libraries. The options which were used to draw the shapes are given by ‘`{circuit, logic <style>, tiny circuit symbols, every circuit symbol/.style={fill=white, draw}}`’, where `<style>` is ‘IEC’, ‘CDH’, or ‘US’. In addition the option ‘`logic gate IEC symbol color=black`’ was used in combination with ‘IEC’.

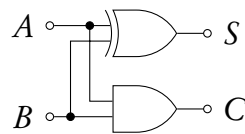
Figure 5.30 is the first example about drawing a logical circuit with `tikz`. There are two new concepts in this example. The first new concept is the option ‘`circuit declare symbol`’ option, which is used to define names of new circuit symbols. The second new concept is the option ‘`set <symbol> graphic`’ option, which is used to define the appearance of the circuit symbol `<symbol>`. The example uses these two options to define two new symbols called ‘`connection`’ and ‘`io`’. The former is used for drawing connections as black filled circles. The latter for drawing the input and output nodes as circles. The style of the remaining symbols in this example determined by the ‘`circuit logic CDH`’ option. Notice that this examples requires the `tikz` library `circuits.logic.CDH`. Finally, note that if you’re drawing many circuits then you can use the `\tikzset` command to define the defaults for your circuits.

## 5.21 Installing `tikz`

This temporary section describes how to obtain a recent `tikz/pgf` distribution and install it.

The easiest way to obtain a recent distribution is going to <http://sourceforge.net/projects/pgf> and saving a recent build. You install the distribution as a normal `TEX` package. This is best done by installing it locally in a dedicated directory called `$(HOME)/$(LATEX)/$(STYLES)/` (or equivalent) which hosts all your private `TEX` style and class files. Section 15.3 describes how this is done.

**Figure 5.30**  
Drawing a half adder with  
tikz.




---

```

\begin{tikzpicture}
  [circuit logic CDH,
   circuit declare symbol=connection,
   circuit declare symbol=io,
   set connection graphic={fill=black,
                           shape=circle,
                           minimum size=1mm},
   set io graphic={draw,shape=circle,
                   minimum size=1mm},
   every circuit symbol/.style={fill=white,draw}]
\draw
  +(0,-1) node[xor gate] (x) {}
  (x.input 1) +(-0.8,0) node[io] (A) {}
  (A |- a.input 2) node[io] (B) {}
  (x.output) -- +(0.4,0) node[io] (S) {}
  (a.output) -- (a.output -| S)
  node[io] (C) {}
  ($ (B)!0.33!(a.input 2)$) node[connection] {}
  |- (x.input 2)
  ($ (A)!0.66!(x.input 1)$) node[connection] {}
  |- (a.input 1)
  (A.west) node[anchor=east] {$A$}
  (B.west) node[anchor=east] {$B$}
  (S.east) node[anchor=west] {$S$}
  (C.east) node[anchor=west] {$C$}
  (A) -- (x.input 1)
  (B) -- (a.input 2);
\end{tikzpicture}

```

---

# CHAPTER 6

## Presenting Data with Tables

THIS CHAPTER STUDIES how to present data using tables. Section 6.1 starts by explaining the purpose of tables. Section 6.2 continues by studying the different kinds of tables. Section 6.3 shows the components of a table. Section 6.4 presents some guidelines about table design. Section 6.5 explains L<sup>A</sup>T<sub>E</sub>X's table environment, which is usually used to present tables. Multi-page tables are studied in Section 6.7. Section 6.8 concludes this chapter by providing some informations about packages providing database and spreadsheet interfaces.

The first part of this section is based on [Bigwood and Spore, 2003]. A L<sup>A</sup>T<sub>E</sub>X view on presenting tables may be found in [Voß, 2008].

### 6.1 The Purpose of Tables

Tables are a common way to communicate facts in newspapers, reports, journals, theses, and so on. There are several advantages of using tables.

- Tables list numbers in systematic fashion.
- Tables supplement, simplify, explain, and condense written material.
- Well-designed tables are easily understood.
  - Patterns and exceptions can be made to stand out.
  - They are more flexible than graphs. For example, in a graph it may be difficult to mix numeric information about data in different units such as the total consumption of petrol in Ireland in tons in the years 1986–2008 year and the average number of rainy days per year in the same country and the same period of time.

### 6.2 Kinds of Tables

There are two kinds of tables: *demonstration tables* and *presentation tables*. The following explains the difference between the two.

**Demonstration tables** In demonstration tables figures are organised to show a trend or show a particular point. Examples are: (most) tables in reports and tables (shown) in meetings.

**Figure 6.1**

This figure shows the components of a typical demonstration table. The background of the table is coloured grey. The black-on-white text to the right of the table describes the components of the table.

Table 3.1. GP and diabetic services, 2000				} Number and Title
Towns	Number	Number providing diabetic services	GP Practices % Providing diabetic services	} Column Headings
Town A	40	38	95	} Row Headings
Town B	29	27	93	
Town C *	29	25	86	
Town D	34	29	85	
Town E	36	30	83	
Town F	<u>62</u>	<u>32</u>	<u>52</u>	
Total	230	181	82	
Source: Health Authority annual Report, 2001				} Source
* Including Town E and Town F.				} Footnote

**Reference tables** Reference tables provide extra and comprehensive information. Examples are: train schedules and stock market listings.

## 6.3 The Anatomy of Tables

Figure 6.1 depicts a typical presentation table, which is based on [Bigwood and Spore, 2003, Page 27]. The table has several components.

**Number and title** In this example, the number and the title are listed at the top of the table. You may also find them at the bottom. The title should describe the purpose of the table. The table's number is used to refer to the table further on in the text. In addition it helps you find the table when you are looking it up.

There are also two other styles of tables. In the first you will find a separate *legend*, which is a description of what is in the table. In the other style, which is the default in L<sup>A</sup>T<sub>E</sub>X, tables have *captions*, which are a combination of number, title, and legend. Good captions should provide a number, a title, and a short explanation of the data listed in the table.

If you include a table, you should always discuss it in the text.

- If the table *is* relevant, does have a message, but is not referred to in the text, then how are you going to draw your reader's attention to the table? After all, you would want your reader to notice the table.
- If the table is *not* relevant to the running text, then why put it in?

**Table 6.1**  
A poorly designed table.

Chilled Meats	Calories
Beef (4 oz/100 g)	225
Chicken (4 oz/100 g)	153
Ham (4 oz/100 g)	109
Liver sausage (1 oz/25 g)	75.023
Salami (1 oz/25 g)	125

- If you don't discuss a table in the running text, then this may confuse and irritate the reader as they may waste a lot of time trying to find where this table is discussed in the text.

**Column headings** The column headings are used to describe the data in the table. In this example, there is a multi-column heading. Horizontal lines separate the column headings from the number and title and from the row heading of the table.

**Row headings** The row headings are the meat of the table. They present the facts, patterns, trend, and exceptions in terms of numbers, and percentages.

**Trend** In this table the general pattern is presented in the last column. Generally, in most towns more than 80% of the GPs provide diabetic services.

**Exception** In this table underlining is used to highlight an exception of the general trend in the table. Other techniques for highlighting exceptions are using a different style of text (bold, italic, ...). However, notice that using different colours to highlight exceptions may not always be a good choice. For example, the difference may not be clear when the table is printed on a black-and-white printer. In addition it may be difficult to reproduce colours with photocopying.

**Source** The source describes the base document from which the table is obtained or is based on.

**Footnote** The footnote provides an additional comment about some of the data.

## 6.4 Designing Tables

This section provides some rules of thumb for the design of tables. To start, consider Table 6.1, which is based on [Bigwood and Spore, 2003, Page 18]. It should be clear that this is a very poorly designed table. There are several things which are wrong with this table. The following are but a few.

- The gridlines make it difficult to scan the data in the table.

- The vertical alignment makes it difficult to compare the numbers in the table.
- The numbers have different widths. This makes it difficult to compare them — even with proper vertical alignment.
- It is not clear what quantities of meat are compared in the last column. This makes it impossible to see the trend of calories per unit of weight. It is possible to work this out from the data in the first column, but this makes life more difficult for the reader. For example, for beef, chicken, and ham, the calories are listed for 4 oz/100 g units. For liver sausage and salami they are listed for 1 oz/25 g units. This is a common error: the information is there but it is poorly presented. As a result the table is useless.
- The precision of the data in the last column is different for different items. For example, for salami, it is listed with three decimals. It is not clear why this is important and it only makes it more difficult to see the trend.

To improve the table we do the following.

- We reorder the information to the same unit: 100 g (4 oz). This allows us to simplify the first column. In addition it is now clear what is listed in the last column.
- We reorder the rows to highlight the trend in the last column.
- We reduce the grid lines to a minimum. This makes it easier to scan the data.
- We present all numbers using the same precision and a similar number of digits. It is now easier to compare the numbers.
- We align the items in the first column to the left. This now makes it easier to scan the items in the first column.
- We align the numbers to the right. This now makes it easier to see the relative differences of the data.
- We use non-proportional numbers. Most  $\text{\LaTeX}$  classes and styles already give you non-proportional numbers. If they don't then they may provide a command to switch to a style with non-proportional numbers. You can always get non-proportional numbers by using the command `\texttt`.
- Optional: make the Column Headings stand out by typesetting them in bold face.

The result, which is listed as Table 6.2, is a much better table.

The main rule of thumb in the design of tables is to keep them simple. Less is more.

- Good tables are simple and uncluttered.
  - The number of vertical grid lines should be reduced to the absolute minimum. The advantage is that it makes it easier to scan the data in the table.



Table 6.2

An improved version of Table 6.1.

Chilled Meats	Calories per 100 g (4 oz)
Salami	500
Liver sausage	300
Beef	225
Chicken	153
Ham	109

- Other gridlines should be kept to a minimum. Arguably, gridlines should only be used to separate (1) the table from the surrounding text, (2) the number and title, (3) the column headings, and (4) the row headings.
- Unless there is a good reason, you should align numbers and column headings to the right. This results a more uniform appearance which makes it easier to compare numbers.
- Good table titles should be concise, definitive, and comprehensive. Where appropriate they should inform the reader about the following.

**What** Table titles should describe the subject of table. For example: Annual income.

**Where** If needed, they should describe the location of the data.

**When** If needed, they should describe the relevant time. This should be kept short: 2000, 1900–1940, May, ....

**Units** If units are used they should be described. Do not mix units, e.g., kilograms and pounds, since this makes it difficult to compare. Instead convert them to the same unit (preferably, International System of Units (SI) units).

- Numbers should be aligned to *facilitate comparison*. For most reference tables and all presentation tables:
  - Numbers should be typeset in a monospaced font.
  - Whole integral numbers should be aligned to the right.
  - Decimal numbers should be aligned to the decimal point.
  - Scaling should be considered if the relative size of the numbers varies much. If this is the case then you should consider converting numbers to thousands, millions, and so on. Alternatively, you should consider using scientific notation:  $\$1.4 \times 10^{+4}\$$  and  $\$2.3 \times 10^{-3}\$$ . Notice that the use of the exponent may disrupt the normal inter-line spacing. Should this be the case then you may also consider using `\texttt{1.4E+4}` and `\texttt{2.3E-3}` or `\texttt{1.4E\, +4}` and `\texttt{2.3E\, -3}`. If all signs of the exponent parts are nonnegative then they may be omitted.

**Figure 6.2**  
Creating a table with the  
booktabs package.

---

```

\begin{table}[tbp]
  \begin{tabular}{ll}
    \toprule
      \textbf{Chilled Meats}
      & \textbf{Calories per} \\
      & \textbf{100\,$\mathrm{g}$/$4\,$\mathrm{oz}$} \\
    \midrule
      ...
    \bottomrule
  \end{tabular}
  \caption[Calories of chilled meats.]
    {Calories of chilled meats per weight. ...}
  \label{tab:meat}}
\end{table}

```

---

- Reduce the amount of whitespace per line. This makes it easier to quickly scan the lines in the tables from left to right. With long lines and much whitespace this is much more difficult. In Table 6.2 the distance between the last letters in the first column and the first numbers in the second column is relatively small. Had we typeset the column heading of the second column on a single line — as opposed to using two lines — the distance would have been greater, leading to a less-quality table.
- For long tables, you should consider adding extra linespace at regular intervals (for example after each fourth or fifth line).

## 6.5 The table Environment

We've already seen how to present tabular information. The `table` environment creates a *floating* table. As with the `figure` environment, this puts the body of the environment in a numbered table, which may be put on a different place in the document than where it's actually defined. The table placement is controlled with an optional argument. This optional argument works as with the optional argument of the `figure` environment. (See Section 4.1 for further information about the optional argument and how it affects the positioning of the resulting table. Inside the table, `\caption` defines a caption, which also works as with `figure`. It is recalled that moving arguments inside captions have to be protected. This is explained in Section 10.2.3. The starred version of the environment (`table*`) produces an unnumbered table, which is not listed in the list of tables.

Figure 6.2 shows how to create a table, which assumes the `booktabs` package is included.

## 6.6 Wide Tables

Sometimes tables may be too wide for the current page. Should this happen the `rotating` package may come to rescue. The package defines a number of commands and environments which are used to implement a `sidewaystable` environment, for presenting rotated tables. The following command typesets  $\langle \text{stuff} \rangle$  in a rotated table.

```
\begin{sidewaystable}
  \langle stuff \rangle
\end{sidewaystable}
```

*LaTeX Usage*

Inside  $\langle \text{stuff} \rangle$ , the command `\caption` works as usual. The rotating package also defines a `sidewaysfigure` environment for figures.

## 6.7 Multi-page Tables

The `longtable` package defines an environment called `longtable`, which has a similar functionality as the `tabular` environment. The resulting structures can be broken by  $\text{\LaTeX}$ 's page breaking mechanism.

Since a single `longtable` may require several page breaks, it may take several runs before it is fully positioned. The `\caption` command works as usual inside the body of a `longtable`.

The `longtable` package needs to know how to typeset the first column heading, subsequent column headings, what to put at the bottom of the table on the last page, and what to put at the bottom of the first pages. This is done with the following commands.

`\endfirsthead`

This indicates the end of the first column headings specification. The material starting at the body of the `longtable` and ending at the command `\endfirsthead` is used to typeset the first column headings. □

`\endhead`

This indicates the end of the specification for remaining column headings. All the material in between `\endfirsthead` and `\endhead` is used to typeset the remaining column headings. □

`\endfoot`

This indicates the end of the specification for remaining column headings. All the material in between `\endhead` and `\endfoot` is used to typeset the bottom of tables on the first pages. □

`\endlastfoot`

This indicates the end of the last foot specification. All the material between `\endfoot` and `\endlastfoot` is used to typeset the bottom of the table on the last page. □

Figure 6.3 demonstrates how to use the `longtable` environment to typeset a table about the nutritional values of chilled meats.

**Figure 6.3**

Using the `longtable` package. The `\newcommand` command at the top of the listing defines a user-defined command called `\boldmc` which works just as the `\multicolumn` command, except for the fact that it typesets the text in a bold face font. User-defined commands are explained in Chapter 10.

---

```

\newcommand{\boldmc}[3]{
  \multicolumn{#1}{#2}{\textbf{#3}}%
}
\begin{longtable}{lr}
  \toprule
  \textbf{Meats}
  & \boldmc{1}{1}{Calories per $100\,,\,\mathrm{g}$}
  \\ \midrule
\endfirsthead
  \toprule
  \boldmc{2}{c}{\tablename~\thetable\ Continued}
  \\ \midrule
  \textbf{Meats}
  & \boldmc{1}{1}{Calories per $100\,,\,\mathrm{g}$}
  \\ \midrule
\endhead
  \midrule
  \boldmc{2}{1}{Continued on next page}
  \\ \bottomrule
\endfoot
  \bottomrule
\endlastfoot
  Salami           & 500
  \\ Liver sausage & 300
  :
\end{longtable}

```

---

## 6.8 Databases and Spreadsheets

There are several packages which let you create and query databases and typeset the result as a table or tabular. Some of these packages provide additional functionality which lets you create barcharts, piecharts, and so on. The following are some of these packages.<sup>1</sup>

**datatool** The `datatool` package [Talbot, 2007] is a very comprehensive package. The package lets you create databases, query them, and modify them. Finally, the package lets you create pie and bar charts or line graphs. Further information may be found in the package documentation [Talbot, 2007].

**pgfplotstable** The `pgfplotstable` package [Feuersänger, 2008], which uses `pgfkeys`, lets you read in tab-separated data and typeset the data as a tabular. The package also supports a limited form of queries. The package lets you round numbers up to a specified precision.

---

<sup>1</sup>At the time of writing this section I haven't had much time to play with these packages. As a consequence the presentation may not be entirely accurate.

**calctab** The `calctab` package [Giacomelli, 2009] lets you define rows in the table with commands. In addition it provides commands which allow you accumulate sums of entries in given columns, and so on. The package documentation is not very long and uses simple examples. Unfortunately the package does not seem to have a facility to set the symbol for the decimal point.

**spreadtab** As the name suggests, the `spreadtab` package [Tellechea, 2010] is written with a spreadsheet in mind. The user specifies a matrix of cells (the spreadsheet) in some form of tabular-like environment. The layout of the matrix determines the result and cells can be rules for computing results from other cells. The package provides a command for setting the symbol for the decimal point.

If all you need is a simple way to compute sums/averages from rows and columns then you should consider using the `spreadtab` package.



# CHAPTER 7

## Presenting Data with Graphs

THIS CHAPTER, which is still in its infancy, studies the presentation of data with graphs with  $\text{\LaTeX}$ . The presentation of this chapter is example driven and mixes theory of presentation with practice. The theory is mostly based on [Bigwood and Spore, 2003]. With the exception of pie charts, which are discouraged anyway, all graphs are created with the recently released `pgfplots` package [Feuersänger, 2010], which creates astonishingly beautiful graphs in a consistent style with great ease. The remainder of this chapter covers pie charts, bar graphs, paired bar charts, component bar graphs, line graphs, and scatter plots but (for the moment?) it excludes 3-dimensional graphs.

### 7.1 The Purpose of Graphs

A picture can say more than a thousand words. This, to some extent, epitomises graphs: good graphs tell a story which is easily recognised and will stick. Graphs are good at showing global relationships, differences, and change.

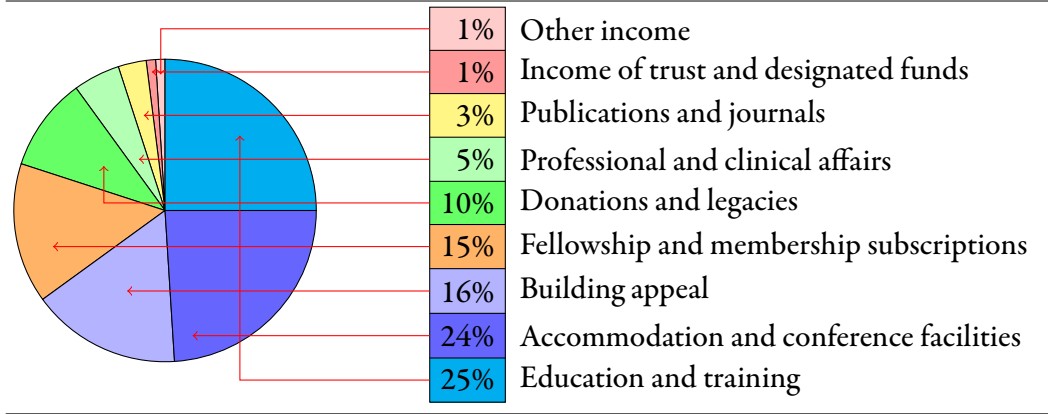
**Global relationships** Graphs are good at showing global relationships. A 2-dimensional scatter plot, for example, may reveal that the data are clustered, that the  $y$ -coordinates have a tendency to increase as the  $x$ -coordinates increase, that most  $x$ -coordinates are smaller than the  $y$ -coordinates, and so on.

**Differences** Graphs are good at showing the difference between two or several functions/trends. For example, the difference between the height of males in two countries as a function of their age, the difference of the running time between four algorithms as a function of the size of the input, and so on.

**Change** Graphs are also good at showing the rate of change within a single function/trend. For example, the rate of change of the running time of a single algorithm as the size of the input increases. Interestingly, differences and change can often be shown effectively in a single graph.

A well-designed graph sticks and conveys the essence of the relationship.

Figure 7.1  
A pie chart.



## 7.2 Pie Charts

Our first kind of graph is the *pie chart*. Pie charts have become very fashionable. Programs such as *excel* have made creating pie charts relatively easy. Figure 7.1 depicts a typical pie chart. The information in the pie chart is adapted from [Bigwood and Spore, 2003]. For sake of this example, the percentages are listed as part of the legend information. Even with this information it is difficult to relate the segments in the chart with the items in the legend.

The relative size of each segment in the pie chart is equal to the relevant size of the contribution of its “label”. To create the chart, the segments are ordered from small to large and presented counter-clockwise, starting at 90°. Colours are usually used to distinguish the segments. Note that care should be taken when selecting colours as they may not always print well.<sup>1</sup> Hatch patterns are avoided as they are distractive. The pie chart in Figure 7.1 has 9 segments, which is too much: good pie charts should have no more than 5 segments [Bigwood and Spore, 2003].

Note that without the percentages it is impossible to compare the relative sizes/contributions of the two smaller segments, which happen to have the same size. Likewise it is difficult to compare the sizes of the segments which contribute 15% and 16% of the total. Arguably, a table is a better way to present the data. As a matter of fact, the legend already is some form of table.

Pie charts are very popular, especially in “slick” presentations. This is surprising as it is well known that pie charts are not very suitable for communicating data and that specialists avoid them [Bigwood and Spore, 2003] (see also the discussion about pie charts in [Tantau, 2010]). Bar graphs, which are studied in the following section, are almost always more effective than pie charts. Despite these observations, pie charts

<sup>1</sup>Note that with careful planning you should be able to change the colours of the segments depending on a global “mode” settings in your document. Specifically, this should allow you to select different, proper colours for an online version and a printable version of your document. Techniques for changing colours and other setting which depend on “modes” are studied further on in this book.



are good at showing parts of a whole by percentages, and how a few components may make up a whole [Bigwood and Spore, 2003].

Finally, the pie chart in Figure 7.1 was drawn using raw `tikz` commands and drawing the chart took a long time. Nicola Talbot's `csvpie` provides some support for drawing pie charts but be aware of the arguments against pie charts.

**Figure 7.2**

Using the `axis` environment.

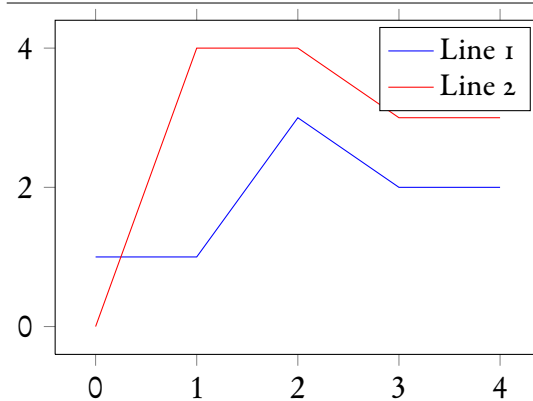
---

```
\begin{tikzpicture}
\begin{axis}[width=8cm, height=6cm, tick align=outside]
\addplot[draw=blue]
    coordinates {(0,1) (1,1) (2,3) (3,2) (4,2)};
\addlegendentry{Line 1}
\addplot[draw=red]
    coordinates {(0,0) (1,4) (2,4) (3,3) (4,3)};
\addlegendentry{Line 2}
\end{axis}
\end{tikzpicture}
```

---

**Figure 7.3**

Output of the `axis` environment in Figure 7.2.



## 7.3 Introduction to `pgfplots`

This section provides an introduction to drawing graphs with the `pgfplots` package which is built on top of `pgf`. The `pgfplots` package lets you draw graphs in a variety of formats. The resulting graphs have a consistent, professional look and feel. The package also lets you import data from `matlab`. As is usual with `pgf` family members, the `pgfplots` manual is impressive.

The workhorse of the `pgfplots` package is an environment called `axis`, which may define one or several *plots* (graphs). Each plot is drawn with the command `\addplot`. When the graphs are drawn the environment also draws a 2- or 3-dimensional axis. The `axis` environment is used inside a `tikzpicture` environment, so you can also use `tikz` commands. Options of the `axis` environment allow you to specify the kind of plot, width, height, and so on. Figure 7.2 demonstrates how to use the command and Figure 7.3 depicts the resulting output.

As is hopefully clear from this example, the options of the `axis` environment set the width to 8 cm, set the height to 6 cm, and force the ticks to be on the outside of the axes.

For reasons of consistency, it is advisable to give the values of the options for your `axis` environments the same value throughout your document. For example, it is very likely that you have a default height and width for your graphs. Ideally, you'd like to define default values for height and width and omit the height and width specifications in the `axis` environment, except when overriding them. This is where the command `\pgfplotsset` comes into play. Basically, `\pgfplotsset` is to `pgfplots` what `\tikzset` is to `tikz`: it lets you set the default values for options of `pgfplots` commands and environments. The following is an example.

```
\pgfplotsset{compat=newest,enlargelimits=0.18, LATEX Input
width=6cm,height=4cm,enlargelimits=0.18}
```

In this example, the command sets the default compatibility to 'newest', which is advised. The default width is set to 6 cm and the default height is set to 4 cm. The spell 'enlargelimits=0.18' increases the default size of the axes by 18%. As with `tikz` commands you may override the values for these options by passing them to the optional arguments of the `axis` environment and the `\addplot` command.

## 7.4 Bar Graphs

Our next graph is the *bar graph*. Bar graphs present quantities as rows or columns. You can use bar graphs to show differences, rates of change, and parts of a whole.

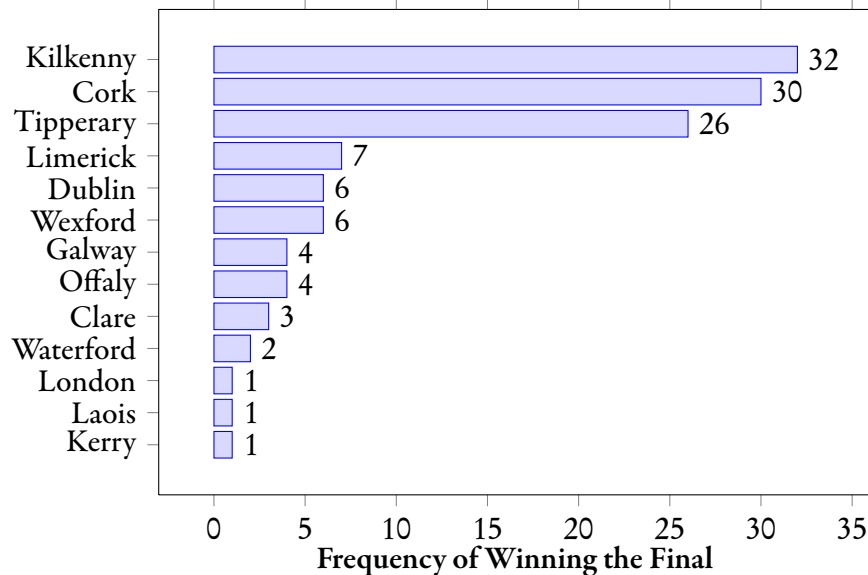
Figure 7.4 depicts a typical bar graph. As with rows in a table, the bars of the graph are ordered to show the main trend. Notice that the data in this graph could just as well have been presented with a table. However, the main advantage of the bar graph presentation is that it "sticks" better. For example, it is very clear from the shape of the bars that Kilkenny, Cork, and Tipperary are the main all-Ireland hurling champions. It is also clear these teams are much better than the rest. With a table, the impact of the difference would not have been so big. Also notice that even in the absence of the frequency information after the bars, it is relatively easy to compare relative differences between the bars.

It is also possible to have bar graphs with vertical bars. Such bar graphs are sometimes used to present changes over time. For example, if you use  $x$ -coordinates for the time, use  $y$ -coordinates for the quantities, and order the bars by time from left to right, then you can see the rate of change of the quantities over time. Of course, you can also present changes over time with horizontal bar graphs but some people find this intuitively less pleasing.

There are at least two reasons why vertical bar graphs are not as

ideal as horizontal bar graphs. First it makes it difficult to label the bars, especially if the text of the labels is long. For example, putting the labels along the  $x$ -axis usually requires rotating the labels, which makes it difficult to read the labels. Second, you can put more bars in a graph with horizontal bars.

**Figure 7.4**  
All-Ireland hurling champions and the number of times they've won the title before 2011.



**Figure 7.5**  
Creating a bar graph.

```
\begin{tikzpicture}
\begin{axis}
[xbar,width=11cm,height=8cm,bar width=10pt,enlargelimits=0.13,
nodes near coords,nodes near co-
ords align=horizontal,
point meta=x * 1, % The displayed number.
legend pos=south east,
xlabel=\textbf{Frequency of Winning the Final},
tick align=outside,
xtick={0,5,...,35}, ytick={1,...,13},
yticklabels={Kerry,Laois,London,Waterford,Clare,Offaly,Galway
Wexford,Dublin,Limerick,Tipperary,Cork,Kilkenny}]
\addplot[draw=blue, fill=blue!15] coordinates
{(1,1) (1,2) (1,3) (2,4) (3,5) (4,6) (4,7)
(6,8) (6,9) (7,10) (26,11) (30,12) (32,13)};
\end{axis}
\end{tikzpicture}
```

Figure 7.5 presents the input which was used to create the bar graph in Figure 7.4. The graph itself is typeset inside an `axis` environment which itself is inside a `tikzpicture` environment. The `xbar` option of the `axis` environment specifies that this is a horizontal bar graph. The data for the graph are provided by the `\addplot` command. The ‘`enlargelimits=0.13`’ option is used to increase the size of the axes by 13%.

The `xtick` and `ytick` keys are used to override the default positions of the  $x$  and  $y$  ticks on the axes. For each `xtick` (`ytick`) position `pgfplots` will place a little tick at the position on the  $x$ -axis ( $y$ -axis) and label the tick with its position. The tick labels can be overridden by providing an explicit list. This is done with the commands `\xticklabels` and `\yticklabels`. The input in Figure 7.5 uses the command `\yticklabels` to override the labels for the  $y$  ticks. The left-to-right order of the labels in the argument of `\yticklabels` is the same as the increasing order of the  $y$  ticks in the bar plot. The command `\xticklabels` works in a similar way.

The lengths of the bars are defined by the required argument of the `\addplot` command. The length of the bar with  $y$ -coordinate  $y$  is set to  $x$  by adding the tuple  $(x, y)$  to the list. For example, the tuple  $(32, 13)$  defines the length of the bar for Kilkenny.

The bar graph in Figure 7.4 also lists the lengths of the bars. These lengths are typeset with the keys `'nodes near coords'`, `'nodes near coords align'`, and `'point meta'`. (By default the lengths of the bars are not typeset.) The `'point meta'` key is used to define the values which are typeset near bars. Since we want to typeset the length of the bar, which is defined by the  $x$ -coordinate in the coordinate list, we use `'x * 1'`. More complex expressions are also allowed.

## 7.5 Paired Bar Graphs

*Paired bar graphs* are like bar graphs but they present information about two groups of data. Figure 7.6 depicts an example of a paired bar graph. In this graph there are two bars for each of the five experiments: one for FC and one for MAC. The information in the graph is made up.

Before studying the  $\text{\LaTeX}$  input which was used to draw the paired bar graph, it should be pointed out that `pgfplots` also lets you construct similar graphs with more than two groups of data. Having pointed this out, it should be noted out that such graphs should be discouraged as the number of bars soon becomes prohibitive, making it difficult to see the trends.

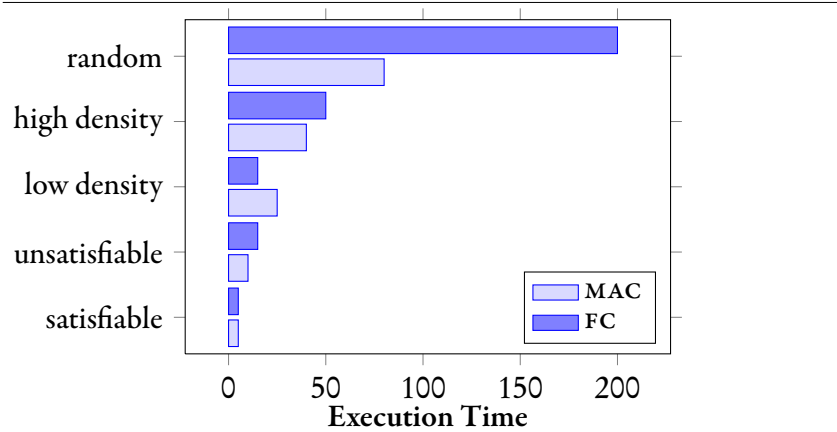
Figure 7.7 shows the input which was used to create the horizontal paired bar graph from Figure 7.6. As you can see from the input, a horizontal paired graph is also created by passing `xbar` as an option to the `axis` environment. The rest of the input is also similar to the input we needed to define our horizontal bar graph. The main differences are that (1) we have two bar classes per  $y$ -coordinate and (2) we have a legend. For each bar class there is an entry in the legend.

Each class of bars is defined by a separate call to `\addplot`. The command `\addlegendentry` adds an entry to the legend for the most recently defined class. The style of the legend entries is set with the `'area legend'` option, which option results in a rectangle drawn in the same way as the corresponding bar. This is slightly nicer than the default legend entry style.

The style of the legend is set with the `legendstyle` key. The ‘`legend pos`’ key is used to position the legend. The spell ‘`cells={anchor=west}`’ aligns the labels of the legend to the left.

**Figure 7.6**

Comparison of the execution time of the Maintain Arc Consistency (MAC) and the Forward Checking (FC) algorithms for instances of 5 problem classes. Execution time in seconds.



**Figure 7.7**

Creating a paired bar graph.

```
\begin{tikzpicture}
\begin{axis}
[xbar,enlargelimits=0.14,width=8cm,height=6cm,,
bar width=10pt,area legend,legend pos=south east,
legend style={legend pos=north east,
cells={anchor=west}},
tick align=outside,xlabel=\textbf{Execution Time},
ytick={1,...,5},
yticklabels={satisfiable,unsatisfiable,
low density,high density,random}]
\addplot[draw=blue,fill=blue!15]
coordinates {(5,1) (10,2) (25,3) (40,4) (80,5)};
\addlegendentry{\textsc{mac}}
\addplot[draw=blue,fill=blue!50]
coordinates {(5,1) (15,2) (15,3) (50,4) (200,5)};
\addlegendentry{\textsc{fc}}
\end{axis}
\end{tikzpicture}
```

## 7.6 Component Bar Graphs

*Component bar graphs*, also known as *stacked bar graphs*, allow you to compare several classes of data. Each class consists of (the same) components and within each class you can see the contribution of the components to the class as a whole. Figure 7.8 depicts a component bar graph.

Notice, again, that the bars are ordered to show the trend. For medal rankings, the first criterion is the number of gold medals won. Ties are broken by considering the number of silver medals, and so on. For other data you may have to order your rows depending on the overall size of

the bars.

For the medal ranking example, it is easy to compare the contribution of the different medals to the overall medal count of a given country. Likewise, it is easy to compare the number of gold, silver, or bronze medals won by different countries. The reason why this works is that all sizes are small and discreet. For different kinds of data, with large ranges of data values, comparing the component sizes is usually not so easy.

Component graphs are usually not the right choice for communicating data, they easily distort data, and the information packed into them is usually too much [Bigwood and Spore, 2003].

Tables may be an interesting and good alternative to component bar graphs. For example, you can have a different row heading for each component in the component graph. If the total size of the bars is important then you can introduce a separate row heading to present these data as the “grand total”, or as the “total time”, and so on.

Figure 7.9 presents the input which was used to create the component bar graph depicted in Figure 7.8. The options ‘`xbar stacked`’ and ‘`stack plots=x`’ indicate that the plot is a horizontal component bar graph. Each `\addplot` command defines the contribution of the next horizontal component for each  $y$ -tick position, so ‘ $(1, 2)$ ’ in the argument of the first `\addplot` command states that the Netherlands (2) won one (1) gold medal. Likewise ‘ $(0, 3)$ ’ in the argument of the second `\addplot` command states that France won no silver medals.

## 7.7 Coordinate Systems

None of our previous `pgfplots`-drawn graphs required additional `tikz` commands for additional lines or text. However, graphs with additional text and lines are quite common. The `pgfplots` package provides several dedicate coordinate systems for correct positioning such additional text and lines. The following are some of these coordinate systems.

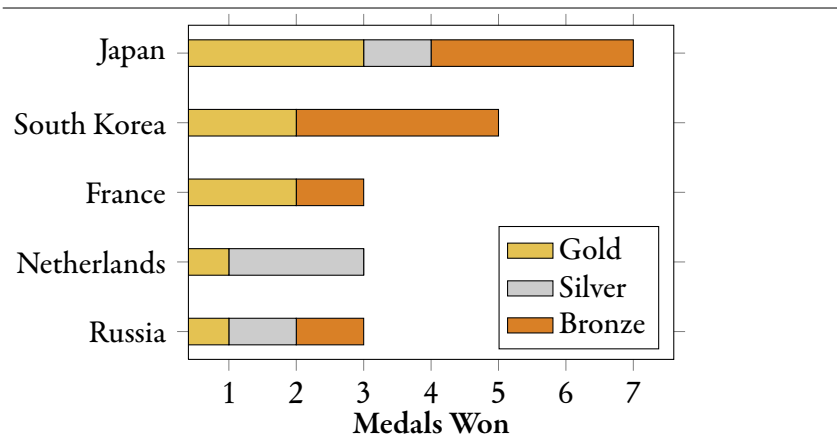
**axis cs** This coordinate system is for “absolute coordinates”. Each coordinate in this system has the same  $x$  and  $y$  coordinates as are used to define coordinates with the `\addplot` command. For example, if you use the command ‘`\addplot{(1,2)(3,4)}`’ then the command ‘`\tikz \draw (axis cs:1,2) node {<text>;}`’ should draw `<text>` at the first coordinate.

**rel axis cs** This coordinate system uses coordinates from the unit square and linearly transforms them to plot coordinates. In this coordinate system the coordinates  $(0, 0)$  and  $(1, 1)$  are the lower left and the upper right corners of the unit square, so ‘`\tikz \draw (rel axis cs:0.5,.5) node {<text>;}`’ should draw `<text>` in the centre of the plot.

**xticklabel cs** This coordinate system is for coordinates along the  $x$ -axis. Basically, the coordinate ‘`xticklabel cs:x`’ is equivalent

**Figure 7.8**

Top five countries of the medal ranking of the 2009 World Judo Championships in Rotterdam (The Netherlands). (Source wikipedia.)

**Figure 7.9**

Creating a component bar graph.

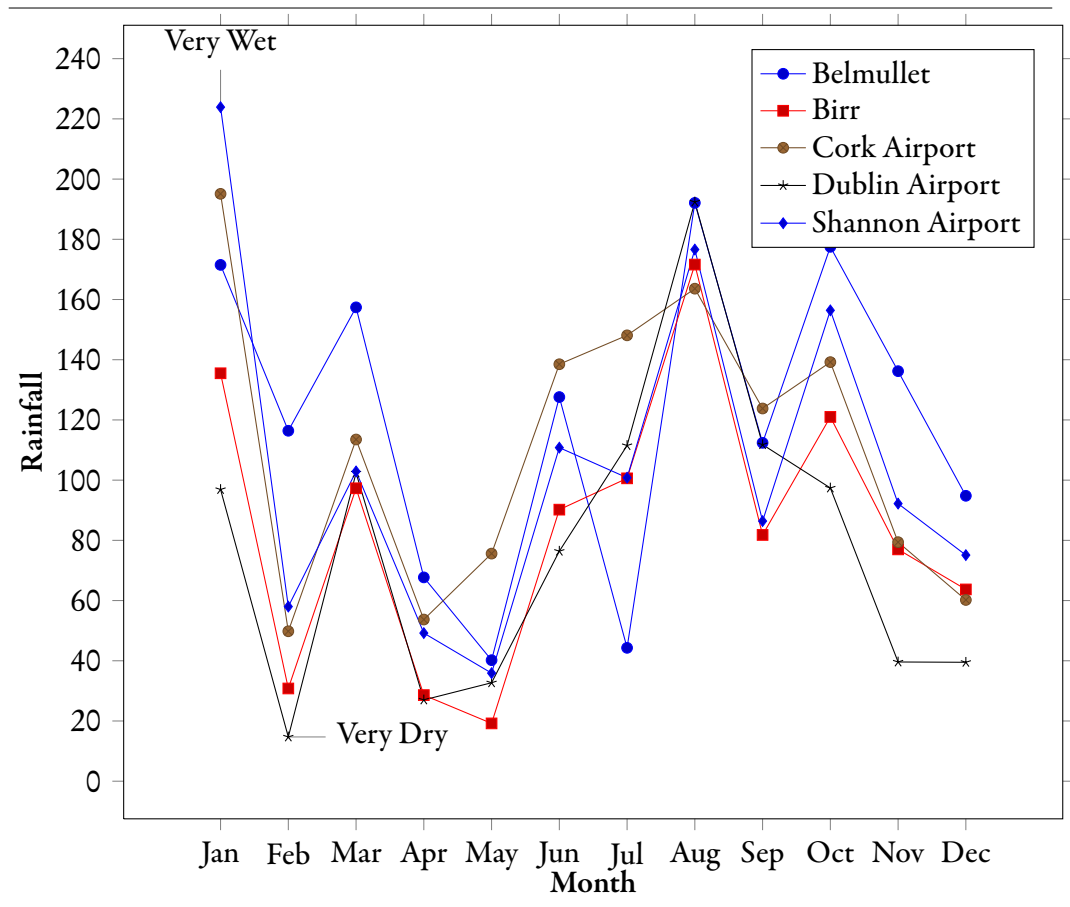
```
\begin{tikzpicture}
\begin{axis}
[xbar stacked, stack plots=x, tick align=outside,
width=8cm, height=6cm, bar width=10pt,
legend style={cells={anchor=west}}, area legend,
xlabel=\textbf{Medals Won}, ytick={1,...,5},
yticklabels={Russia,Netherlands,France,
South Korea,Japan}]
\addplot[draw=black,yellow!50!brown]
coordinates {(1,1) (1,2) (2,3) (2,4) (3,5)};
\addlegendentry{Gold}
\addplot[draw=black,white!60!gray]
coordinates {(1,1) (2,2) (0,3) (0,4) (1,5)};
\addlegendentry{Silver}
\addplot[draw=black,orange!70!gray]
coordinates {(1,1) (0,2) (1,3) (3,4) (3,5)};
\addlegendentry{Bronze}
\end{axis}
\end{tikzpicture}
```

to ‘rel axis cs:  $x, 0$ ’. So far, this is not very interesting. However, the coordinate system also lets you provide an additional coordinate, which should be a length. When provided, the length defines the distance of a shift “away” from the labels on the  $x$ -axis.

**yticklabel cs** This coordinate system is for coordinates along the  $y$ -axis. It works similar to the ‘xticklabel cs’ coordinate system.

The remaining sections provide examples which use some of these coordinate systems. The reader is referred to the `pgfplots` package documentation [Feuersänger, 2010] for further information.

**Figure 7.10**  
Monthly rainfall in millimetres for the year 2009. (Source <http://www.cso.ie>.)



## 7.8 Line Graphs

*Line graphs* are ideal for presenting differences between data sets and presenting the rate of change within individual data sets. They are commonly used to present data (observations) which are a function of (depend on) a given parameter. For example, the running time of a given algorithm as a function of the input size, the average height of males as a function of their age, and so on.

Figure 7.10 depicts a typical line graph. The legend in the top right hand corner of the graph labels the line types in the graph. In general legends should be avoided: if possible the lines should be directly labelled [Bigwood and Spore, 2003], which is to say that each label should be near its line. The main reasons for avoiding legends is that they distract and make it more difficult to relate the lines and their labels. For the graph in Figure 7.10 direct labelling is virtually impossible.

Figure 7.11 depicts the input which was used to create the line graph in Figure 7.10. Most of this is pretty straightforward. The command `\addplot+` is used to define the lines in the graph. The extra plus in the command results in extra marks on the lines for the coordinates in the required argument of `\addplot+`. The option `'sharp plot'` of the `\addplot` command states that consecutive points in the plot should be



**Figure 7.11**  
Creating a line graph.

---

```

\begin{tikzpicture}
\begin{axis}
  [width=\textwidth, enlargelim-
  its=0.13, tick align=outside,
  legend style={cells={anchor=west}, legend pos=north east},
  xticklabels={Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec},
  xtick={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12},
  xlabel=\textbf{Month}, ylabel=\textbf{Rainfall}]
\node[coordinate, pin=above: {Very Wet}]
at (axis cs:1, 223.9) {};
\node[coordinate, pin=right: {Very Dry}]
at (axis cs:2, 14.7) {};
\addplot+[sharp plot] coordinates
  {(1, 171.5) (2, 116.4) (3, 157.4) (4, 67.7) (5, 40.2) (6, 127.6)
   (7, 44.3) (8, 192.1) (9, 112.4) (10, 177.5) (11, 136.2) (12, 94.8)};
\addlegendentry{Belmullet}
\addplot+[sharp plot] coordinates
  {(1, 135.5) (2, 30.8) (3, 97.3) (4, 28.6) (5, 19.2) (6, 90.2
   (7, 100.6) (8, 171.6) (9, 81.8) (10, 121.0) (11, 77.0) (12, 63.7)};
\addlegendentry{Birr}
\addplot+[sharp plot] coordinates
  {(1, 195.1) (2, 49.8) (3, 113.5) (4, 53.7) (5, 75.6) (6, 138.5
   (7, 148.1) (8, 163.6) (9, 123.8) (10, 139.2) (11, 79.4) (12, 60.2)};
\addlegendentry{Cork Airport}
\addplot+[sharp plot] coordinates
  {(1, 96.9) (2, 14.7) (3, 102.4) (4, 27.0) (5, 32.7) (6, 76.4
   (7, 111.5) (8, 192.4) (9, 111.8) (10, 97.4) (11, 39.6) (12, 39.5)};
\addlegendentry{Dublin Airport}
\addplot+[sharp plot] coordinates
  {(1, 223.9) (2, 58.0) (3, 102.9) (4, 49.2) (5, 35.9) (6, 110.8
   (7, 100.8) (8, 176.6) (9, 86.4) (10, 156.4) (11, 92.2) (12, 75.1)};
\addlegendentry{Shannon Airport}
\end{axis}
\end{tikzpicture}

```

---

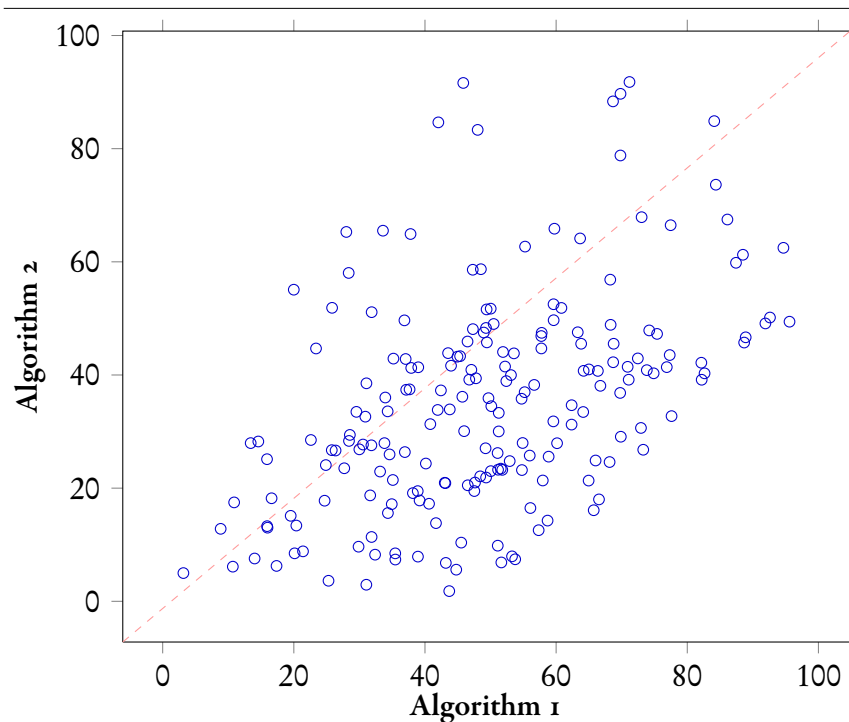
connected using a straight line segment. This is more than likely what you want when you're crating line graphs. The `\node` commands at the end of the `axis` environment draw the texts 'Very Wet' and 'Very Dry' using the 'axis cs' coordinate system. The node shape 'pin' is new but it should be clear how it works.

Finally, notice that line graphs have a tendency to become crowded as the number of lines increases. If this happens you should consider reducing the number of lines in your graph. Alternatively, you may consider using the `spy` feature to zoom in on the important crowded areas in you graph. The `spy` mechanism is explained in Section 5.18.

## 7.9 Scatter Plots

*Scatter plots* are ideal for discovering relationships among a huge/large set of 2-dimensional data points. Basically, the plot has a mark at each coordinate for each data point. Figure 7.12 is a scatter plot which is used to compare the running times of two algorithms for different input. For each input,  $i$ , the scatter plot has a point at position  $(x_i, y_i)$ , where  $x_i$  is the running time of the first algorithm for input  $i$  and  $y_i$  is the running time of the second algorithm for input  $i$ .

**Figure 7.12**  
Running time of Algorithm 1 versus running time of Algorithm 2. Running times in seconds. The majority of the coordinates are below the line  $x = y$ .



**Figure 7.13**  
Creating a scatter plot.

```
\begin{tikzpicture}
\begin{axis}
  [width=\textwidth, tick align=outside,
  xlabel=\textbf{Algorithm~1},
  ylabel=\textbf{Algorithm~2}]
\addplot[scatter,only marks,mark=o,
  draw=blue,scatter src=explicit]
  file {data.dat};
\draw[dashed,red!40] (rel axis cs:0,0) --
  (rel axis cs:1,1);
\end{axis}
\end{tikzpicture}
```

As you can see from the scatter plot, Algorithm 1 usually takes more time than Algorithm 2 for random input. Furthermore, the overall shape of the plot suggests that the running times are positively correlated. The dashed red line helps detecting both trends in the plot.

Table 7.1

This table lists the values for the mark option. The options at the top of the table are standard. The remaining options rely on the tikz library plotmarks.

		Standard	
mark=*		mark=x	
mark=+		mark=-	
With \usetikzlibrary{plotmarks}			
mark=		mark=o	
mark=asterisk		mark=star	
mark=oplus		mark=oplus*	
mark=otimes		mark=otimes*	
mark=square		mark=square*	
mark=triangle		mark=triangle*	
mark=diamond		mark=diamond*	
mark=pentagon		mark=pentagon*	

Figure 7.13 presents the code which was used to create the scatter plot in Figure 7.12. The option ‘scatter’ states that the coordinates provided by the calls to `\addplot` are for scatter plots. The option ‘only marks’ results in a mark which is drawn at each coordinate which is specified by `\addplot`. The style of the mark may be set with the style ‘`mark=<mark style>`’. Possible values for `<mark style>` and the resulting marks are listed in Table 7.1. In our example we’re using the style ‘o’ which results in a circle. The option ‘`color=<colour>`’ sets the colour of the mark.

The option ‘`scatter src=explicit symbolic`’ states that the coordinates are expected as explicit coordinates. Usually scatter plots consist of many data points. Adding all point specifications to the main L<sup>A</sup>T<sub>E</sub>X source of your pgfplots environments surely doesn’t make it easier to maintain the environments. This is why pgfplots provides support for including data from external source files. In our example, ‘`file {data.dat}`’ indicates that the coordinates are in the external file `data.dat`. All lines in this file are of the form ‘`<x-coordinate> <y-coordinate>`’.

The red dashed line is drawn at the end of the axis environment. The ‘`rel axis cs`’ coordinate system is used to specify the start and endpoint of the line. It is recalled from Section 7.7 that this coordinate system scales all coordinates to the unit square with lower left coordinate (0,0) and upper right coordinate (1,1).



## **Part IV**

# **Mathematics and Algorithms**



# CHAPTER 8

## Mathematics

THIS CHAPTER is an introduction to typesetting basic mathematics in  $\text{\LaTeX}$ . For further information the reader is referred to a proper  $\text{\LaTeX}$  book such as [Lamport, 1994], a tutorial such as [Oetiker et al., 2007], or a book on using  $\text{\LaTeX}$  for writing mathematics [Voß, 2009]. A comprehensive listing of  $\text{\LaTeX}$  symbols, including math symbols, is provided by [Pakin, 2005].

$\text{\LaTeX}$ 's basic support for mathematics is limited, which is why the AMS provide a package called `amsmath` which redefines some existing commands and environments and provides additional commands and environments for mathematical typesetting. Throughout this chapter it is assumed that you have installed the `amsmath` package.

The remainder of this chapter is as follows. Section 8.1 starts by describing the  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{\LaTeX}$  software. Section 8.2 describes  $\text{\LaTeX}$ 's ordinary and displayed math mode. Commands for typesetting expressions in ordinary math mode are explained in Section 8.3. Subscripts and superscripts are explained in Section 8.4. Section 8.5 explains how to typeset Greek letters. Environments for typesetting expressions in displayed math mode are explained in Section 8.6. A command for typesetting text inside math expressions is described in Section 8.7. Section 8.8 is dedicated to the task of typesetting delimiters. Commands for typesetting fractions are presented in Section 8.9. Section 8.10 presents commands for typesetting sums, products, and related constructs. Section 8.11 explains the  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\text{\LaTeX}$ 's commands for defining new operator symbols. Section 8.12 continues by presenting commands for integration and differentiation. Section 8.13 explains how to typeset roots. This is followed by Section 8.14 which is dedicated to arrays and matrices. Accents, hats, and other decorations are covered in Section 8.15. Section 8.16 covers overbraces and underbraces. Section 8.17 presents solutions for typesetting case-based definitions. The finer details of typesetting function definitions are explained in Section 8.18. Section 8.19 provides an introduction to the `amsmath` commands for defining and uniform presentation of theorem-like environments. Mathematical punctuation, spacing, and line breaks are covered by Sections 8.20 and 8.21. Section 8.22 explains how to change the type style in math mode. Section 8.23 concludes with tables of useful symbols.

## 8.1 The $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$ Platform

$\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  is a useful platform for typesetting mathematics. The software is provided by the AMS (<http://ams.org/>). The software is freely available and should come with any good  $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  distribution. You can download the AMS software and documentation from <http://www.ams.org/tex/amslatex.html>.

The software distributed under the name  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  consists of various extensions for  $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$ . The distribution is divided into two parts:

**amsc1s** The `amsc1s` class provides the AMS document class and theorem package. Using this class gives your  $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  document the general structure and appearance of an AMS article or book.

**amsmath** An extension package providing facilities for writing math formulas and to improving the typography.

Throughout this chapter  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  and `amsmath` are used interchangeably. The `amsmath` package is really a collection of packages. If you include `amsmath` then you include them all. It lets you to configure some of their basic settings. As usual this is done by passing options inside the square brackets to the `\usepackage` command: `\usepackage[options]{amsmath}`. Some of the options for `amsmath` are as follows:

**leqno** Place equation numbers on the left.

**reqno** Place equation numbers on the right.

**fleqn** Position equations at a fixed indent from the left margin.

Some of the packages provided by  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  are the following. The description of the packages has been adapted from the  $\mathcal{A}\mathcal{M}\mathcal{S}$ - $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  documentation [American Mathematical Society, 2002].

**amsmath** Defines extra environments for multiline displayed equations, as well as a number of other enhancements for math (includes the `amstext`, `amsbsy`, and `amsopn` packages).

**amstext** Provides a `\text` command for typesetting text inside a formula.

**amsopn** Provides `\DeclareMathOperator` command for “operator names” like `\sin` and `\lim`.

**amsthm** Provides a proof environment and extensions for the `\newtheorem` command.

**amscd** Provides an environment for simple commutative diagrams.

**amsfonts** Provides extra fonts and symbols, including boldface (`\mathbf`), blackboard boldface (`\mathbb`), and fractur (`\mathfrak`).

**amssymb** Provides lots of extra symbols.



## 8.2 L<sup>A</sup>T<sub>E</sub>X's Math Modes

L<sup>A</sup>T<sub>E</sub>X has three basic modes which determine how it typesets its input. The basic modes are:

**Text mode** In this mode the output does not have mathematical content and is typeset as text. Typesetting in text mode is explained in Chapter 1.

**Ordinary math mode** In this mode the output has mathematical content and is typeset in the running text. Ordinary math mode is more-commonly referred to as *inline math mode*.

**Display math mode** In this mode the output has mathematical content and is typeset in a display.

The mechanism for typesetting mathematics in ordinary (inline) math mode is explained in the following section. This is followed by some sections explaining some basic math mode typesetting commands, which are then used in Section 8.6. The main purpose of Section 8.6 is to describe some environments for typesetting displayed math.

## 8.3 Ordinary Math Mode

This section explains how to typeset mathematics in ordinary (inline) math mode. It is recalled from the previous section that this means that the resulting math is typeset in the running text. Typesetting in displayed math mode is postponed until Section 8.6. The '\$' operator switches from text mode to ordinary math mode and back, so '\$a = b\$' results in ' $a = b$ ' in the running text. The following provides another example. If you don't understand the constructs inside the '\$ · \$' expressions then don't worry: they are explained further on.

<p>The Binomial Theorem states that</p> $\sum_{i=0}^n \binom{n}{i} a^i b^{n-i} = (a + b)^n.$ <p>Substituting 1 for <math>a</math> and 1 for <math>b</math> this gives us</p> $\sum_{i=0}^n \binom{n}{i} = 2^n.$	L <sup>A</sup> T <sub>E</sub> X Input
---	---------------------------------------

The following is the resulting output. The mathematical expressions in the output are typeset in the running text. This should not come as a surprise since '\$ · \$' is for typesetting in ordinary (inline) math mode.

<p>The Binomial Theorem states that <math>\sum_{i=0}^n \binom{n}{i} a^i b^{n-i} = (a + b)^n</math>. Substituting 1 for <math>a</math> and 1 for <math>b</math> this gives us <math>\sum_{i=0}^n \binom{n}{i} = 2^n</math>.</p>	L <sup>A</sup> T <sub>E</sub> X Output
--	--

## 8.4 Subscripts and Superscripts

Subscripts and superscripts are first-class citizens in mathematics. We've already seen subscripts and superscripts in some of the examples. This section formally explains how to use them.

The superscript operator ( $\wedge$ ) is for creating superscripts. The expression  $\text{\texttt{\$}\langle expr \rangle^\langle sup \rangle\text{\texttt{\$}}}$  makes  $\langle sup \rangle$  a superscript of  $\langle expr \rangle$ . So  $\text{\texttt{\$}e = mc^2\text{\texttt{\$}}}$  gives you  $e = mc^2$ . Grouping works as usual. So to typeset  $e^{a+b}$ , you need braces:  $\text{\texttt{\$}e^{\{a+b\}}\text{\texttt{\$}}}$ .

Subscripts are handled in a similar to superscripts. The subscript operator ( $\_$ ) is for creating subscripts. The expression  $\text{\texttt{\$}\langle expr \rangle_\langle sub \rangle\text{\texttt{\$}}}$  makes  $\langle sub \rangle$  a subscript of  $\langle expr \rangle$ . So to get  $f_{n+2} = f_{n+1} + f_n$ , you need  $\text{\texttt{\$}f_{\{n + 2\}} = f_{\{n + 1\}} + f_{\{n\}}\text{\texttt{\$}}}$ .

Subscripts and superscripts may be nested and combined. The expressions  $\text{\texttt{\$}\langle expr \rangle_\langle sub \rangle^\langle sup \rangle\text{\texttt{\$}}}$  and  $\text{\texttt{\$}\langle expr \rangle^\langle sup \rangle_\langle sub \rangle\text{\texttt{\$}}}$  are equivalent and make  $\langle sub \rangle$  a subscript of  $\langle expr \rangle$  and  $\langle sup \rangle$  a superscript of  $\langle expr \rangle$ , so  $\text{\texttt{\$}s^{\{m + 1\}}_{\{n+2\}}\text{\texttt{\$}}}$  gives you  $s_{n+2}^{m+1}$ .

It is good practice to avoid  $\text{su*su*scripts}$  and  $\text{su*su*su*scripts}$  as much as possible — some style and class files may reject them. The following are some advantages.

**Simplicity** The fewer the  $\text{su*scripts}$ , the simpler the notation, the greater the transparency.

**Readability** The resulting expression is easier to parse.

**Spacing** Avoiding nested subscripts and superscripts reduces the number of inconsistencies in interline spacing.

## 8.5 Greek Letters

This section describes the commands for Greek letters in math mode. These commands do *not* work in text mode.

There are three classes of lowercase letters. The following are the classes and the commands to typeset the letters in the classes.

**Regular** These are the regular lowercase Greek letters. The commands for typesetting these letters are  $\text{\texttt{\backslash alpha}}$  ( $\alpha$ ),  $\text{\texttt{\backslash beta}}$  ( $\beta$ ),  $\text{\texttt{\backslash gamma}}$  ( $\gamma$ ), ....

**Italic** There are also some commands for italic lowercase Greek letters. These commands all start with  $\text{\texttt{\backslash var}}$ :  $\text{\texttt{\backslash varepsilon}}$  ( $\varepsilon$ ),  $\text{\texttt{\backslash vartheta}}$  ( $\vartheta$ ),  $\text{\texttt{\backslash varrho}}$  ( $\varrho$ ), .... These commands are provided by  $\text{\texttt{amsmath}}$ .

**Dunno** Finally there is the  $\mathcal{AMS}\text{-}\text{\texttt{L}\text{A}\text{T}\text{E}\text{X}}$ -provided command  $\text{\texttt{\backslash digamma}}$ , which gives you  $\varphi$ .

There are also commands for uppercase Greek letters. Commands are only provided for letters which are different from the uppercase Roman letters. For example, there is no need for uppercase letters  $A$ ,  $B$ ,  $E$ , and so on. There are two classes of letters:

**Table 8.1**

This table lists the math mode commands for lowercase Greek letters. The commands at the top of the table are standard  $\text{\LaTeX}$  commands. The command  $\backslash\text{digamma}$  and the commands starting with  $\backslash\text{var}$  are provided by  $\mathcal{AMS}\text{-}\text{\LaTeX}$ .

Standard commands		
$\alpha$ $\backslash\alpha$	$\iota$ $\backslash\iota$	$\tau$ $\backslash\tau$
$\beta$ $\backslash\beta$	$\kappa$ $\backslash\kappa$	$\upsilon$ $\backslash\upsilon$
$\gamma$ $\backslash\gamma$	$\lambda$ $\backslash\lambda$	$\phi$ $\backslash\phi$
$\delta$ $\backslash\delta$	$\mu$ $\backslash\mu$	$\chi$ $\backslash\chi$
$\epsilon$ $\backslash\epsilon$	$\nu$ $\backslash\nu$	$\rho$ $\backslash\rho$
$\zeta$ $\backslash\zeta$	$\xi$ $\backslash\xi$	$\psi$ $\backslash\psi$
$\eta$ $\backslash\eta$	$\omicron$ $\backslash\omicron$	$\sigma$ $\backslash\sigma$
$\theta$ $\backslash\theta$	$\pi$ $\backslash\pi$	$\omega$ $\backslash\omega$
$\mathcal{AMS}\text{-}\text{\LaTeX}$ provided commands		
$\varepsilon$ $\backslash\varepsilon$	$\kappa$ $\backslash\kappa$	$\varrho$ $\backslash\varrho$
$\varphi$ $\backslash\varphi$	$\vartheta$ $\backslash\vartheta$	$\varpi$ $\backslash\varpi$
$\varsigma$ $\backslash\varsigma$		
$\digamma$ $\backslash\digamma$		

**Table 8.2**

This table lists the math mode commands for uppercase Greek letters. The commands at the top of the table are standard  $\text{\LaTeX}$  commands. The commands starting with  $\backslash\text{var}$  are provided by  $\text{amsmath}$ .

Standard commands		
$\Gamma$ $\backslash\Gamma$	$\Xi$ $\backslash\Xi$	$\Phi$ $\backslash\Phi$
$\Delta$ $\backslash\Delta$	$\Pi$ $\backslash\Pi$	$\Psi$ $\backslash\Psi$
$\Theta$ $\backslash\Theta$	$\Sigma$ $\backslash\Sigma$	$\Omega$ $\backslash\Omega$
$\Lambda$ $\backslash\Lambda$	$\Upsilon$ $\backslash\Upsilon$	
$\mathcal{AMS}\text{-}\text{\LaTeX}$ provided commands		
$\Gamma$ $\backslash\varGamma$	$\Xi$ $\backslash\varXi$	$\Phi$ $\backslash\varPhi$
$\Delta$ $\backslash\varDelta$	$\Pi$ $\backslash\varPi$	$\Psi$ $\backslash\varPsi$
$\Theta$ $\backslash\varTheta$	$\Sigma$ $\backslash\varSigma$	$\Omega$ $\backslash\varOmega$
$\Lambda$ $\backslash\varLambda$	$\Upsilon$ $\backslash\varUpsilon$	

**Regular**  $\backslash\Gamma$ ,  $\backslash\Delta$ ,  $\backslash\Theta$ , .... These commands are standard  $\text{\LaTeX}$ .

**Italic**  $\backslash\varGamma$ ,  $\backslash\varDelta$ ,  $\backslash\varTheta$ , .... These non-standard commands are provided by  $\mathcal{AMS}\text{-}\text{\LaTeX}$ .

Table 8.1 lists the commands for the lowercase and Table 8.2 lists the commands for the uppercase Greek letters.

## 8.6 Displayed Math Mode

This entire section is dedicated to displayed math material. Standard  $\text{\LaTeX}$  provides a few commands for displayed math. The  $\text{amsmath}$  package redefines some of them and provides several extensions. As usual unstarred versions of the environments are numbered in the text. Starred versions are not numbered.

Some environments allow vertical alignment in multi-line expressions. In such environments linebreaks and vertical alignment are specified as follows.

- Vertical alignment positions are specified with &.
- Line breaks are specified with \\.

The unstarred versions of the environment produce labels: `equation`, `align`, .... The starred versions of the environment do not produce labels: `equation*`, `align*`, ....

As a note of advice, you should avoid the unstarred versions if there are no references to the equations in the text. If you decide otherwise, the following may happen. A reader may notice an equation's label. They may start looking for the text that refers to the label. (They're trying to find additional information.) They may not be able to find the text location. (After several attempts!) They may get confused and irritated. If the reader is a referee this may be the final drop which was needed for them to reject your paper.

The remainder of this section consists of examples of some of `ams-math`'s displayed equation environments. All examples use the unstarred versions.

### 8.6.1 The `equation` Environment

The `equation` environment is for typesetting a *single* numbered displayed equation. It is one of the more important environments for typesetting displayed math material.

The following demonstrates how to use the environment. The example uses a few new commands which are explained in more detail further on. A short description of the commands is as follows. The `\sum` command is for typesetting sums. The subscript (`_`) and superscript (`^`) commands are used to specify the lower and upper limits of the index variable in the summand. The command `\sum` is explained in detail in Section 8.10. The command `\binom` is for typesetting binomial coefficients. The *thin space command* (`\,`) is used to generate a thin space just before the period in the display.

L<sup>A</sup>T<sub>E</sub>X Input

The following is Newton's Binomial Theorem:

```
\begin{equation}
  \label{eq:Newton}
  \sum^{n}_{i=0} \binom{n}{i} a^i b^{n-i} = (a+b)^n \,,
\end{equation}
```

Substituting  $\$1\$$  for  $\sim \$a\$$  and for  $\sim \$b\$$  in  $(\ref{eq:Newton})$  gives us  $\$ \sum^{n}_{i=0} \binom{n}{i} = 2^n \$$ .

The following is the resulting output. Notice that the display makes the equation stand out clearly from the surrounding text. This is the main purpose of the display.

The following is Newton's Binomial Theorem:

LaTeX Output

$$\sum_{i=0}^n \binom{n}{i} a^i b^{n-i} = (a+b)^n. \quad (8.1)$$

Substituting 1 for  $a$  and for  $b$  in (8.1) gives us  $\sum_{i=0}^n \binom{n}{i} = 2^n$ .

Also notice that the equation in the output is automatically numbered and that the labelling and referencing mechanism in the input is standard:

- You may define a label for the number of the equation with the `\label` command.
- Once the label is defined, you get the number of the equation by applying the `\ref` command to the label. In the previous example we put the equation number inside parentheses: ‘`(\ref{eq:Newton})`’. This is a common way of referring to equations. Arguably it is better to use the `prettyref` package when typesetting references. This is explained in Section 1.5.

There is also a starred version (`equation*`) of the `equation` environment. As is the default this results in an unnumbered version of its unstarred equivalent. LaTeX also has a different mechanism for typesetting a single unnumbered equation. The command `\[` starts such equations and the command `\]` ends them.

### 8.6.2 The `split` Environment

The `split` environment is for splitting a *single* equation into several lines. The environment allows you to align the resulting equation. The environment cannot be used at the top level and can only be used as part of (some of the) other `amsmath` environments such as `equation` and `gather`. The `split` environment does not number the resulting equation. The following shows how to use the environment.

```
\begin{equation}
\begin{split}
a &= b + c + d && \\
&\quad + f + g + h && \\
&> 0, . &&
\end{split}
\end{equation}
```

LaTeX Input

The following is the resulting output. As you can see from the input and the resulting output, the position of the vertical alignment is indicated using the alignment operator (`&`) in the input and linebreaks are forced with the newline operator (`\\`). This is the default mechanism for specifying vertical alignment and linebreaks. The vertical alignment position is just to the left of the equality symbol. This is why the line starting with a plus is indented a bit. This is done with the command

`\qqquad`, which generates a horizontal space which is roughly equivalent to twice the width of the uppercase letter M. Section 8.2.1.1 provides more information about how to use the command `\qqquad`.

*LaTeX Output*

$$\begin{aligned} a &= b + c + d \\ &\quad + f + g + h \\ &> 0. \end{aligned} \quad (8.2)$$

As mentioned before, `split` does not number its output. This explains why there is no starred version of `split`. In the output of the previous example, the numbering of the equation is controlled by the `equation` environment, which numbers the output which is created by `split`.

### 8.6.3 The `multline` Environment

The `multline` environment is for displaying a *single* equation over multiple lines. The environment does *not* allow control with vertical alignment. The resulting output gets only one number. There is also a starred version of the `multline` environment.

The lines are typeset as follows:

**First line** The first line is aligned to the left.

**Last line** The last line is aligned to the right.

**Middle lines** The remaining lines are aligned to the centre. However, the `\shoveleft` command may be applied to move these lines to the left. The command can only be used inside the `multline` environment. Likewise, the command `\shoveright` can be used to force lines to the right.

The reader is referred to the `amsmath` documentation [American Mathematical Society, 2002] for further information. The following demonstrates how to use the environment.

*LaTeX Input*

```
\begin{multline}
  a = b + c + d \\
  + f + g + h \\
  \shoveleft {+ k + l + m} \\
  + n + o + p \,.
\end{multline}
```

The output looks like this:

*LaTeX Output*

$$\begin{aligned} a &= b + c + d \\ &\quad + f + g + h \\ + k + l + m &\quad + n + o + p. \end{aligned} \quad (8.3)$$

#### 8.6.4 The gather Environment

The `gather` environment is for displaying a group of consecutive equations *without* vertical alignment. All resulting equations are numbered and centred. The environment also has a starred version.

The following example demonstrates how to use the environment.

```
\begin{gather}
    a = b\,, \qquad \\\
    \begin{split}
        a &= m + n + o \qquad \\
        &\quad \&\ \mathrm{qqquad} + x + y + z \ , .
    \end{split}
\end{split}
\end{gather}
```

The following is the resulting output. Again notice that the equation which is constructed using the `split` environment occupies two lines in the resulting output but only receives one number.

$$a = b, \quad (8.4)$$

$$a = m + n + o \quad (8.5)$$

$$+ x + y + z.$$

### 8.6.5 The align Environment

The `align` environment is for equation groups with mutual vertical alignment. Each row is numbered separately. The command `\nonumber` turns off the numbering of the current equation.

There is also a starred version of the `align` environment. The following shows how to use the unstarred version of the environment. In this example the command `\infty` is for typesetting the infinity symbol ( $\infty$ ).

```

\begin{align}
&\label{eq:one}
F &= \sum^{\infty}_{n=0} f_n z^n && \\\
&\label{eq:two}
&= z + \sum^{\infty}_{n=2} (f_{n-1} + f_{n-2}) z^n && \\\
&\label{eq:three}
&= z + F/z + F/z^2 && \\\
&\nonumber
&= z / (1 - z - z^2) && \backslash, .
\end{align}

```

Here the last equation is obtained from~(\ref{eq:one}),  
(\ref{eq:two}), and~(\ref{eq:three}) by transitivity  
of equality and by solving for~\$F\$.

The following is the resulting output. Notice that the \nonumber in the input suppresses the number of the corresponding equation/row

in the output. The labels of the remaining three equations are defined as usual and are used in the text following the display to refer to the numbered equations.

LaTeX Output

$$F = \sum_{n=0}^{\infty} f_n z^n \quad (8.6)$$

$$= z + \sum_{n=2}^{\infty} (f_{n-1} + f_{n-2}) z^n \quad (8.7)$$

$$= z + F/z + F/z^2 \quad (8.8)$$

$$= z/(1 - z - z^2).$$

Here the last equation is obtained from (8.6), (8.7), and (8.8) by transitivity of equality and by solving for  $F$ .

The `align` environment also allows you to have more than one column. The following shows how this is done.

LaTeX Input

```
\begin{align}
a_0 &= b_0 \backslash, , & b_0 &= c_0 \backslash, , & c_0 &= d_0 \backslash, , \backslash \\
a_1 &= b_1 \backslash, , & b_1 &= c_1 \backslash, , & c_1 &= d_1 \backslash, , \backslash \\
a_2 &= b_2 \backslash, , & b_2 &= c_2 \backslash, , & c_2 &= d_2 \backslash, . \\
\end{align}
```

The following is the resulting output.

LaTeX Output

$$a_0 = b_0, \quad b_0 = c_0, \quad c_0 = d_0, \quad (8.9)$$

$$a_1 = b_1, \quad b_1 = c_1, \quad c_1 = d_1, \quad (8.10)$$

$$a_2 = b_2, \quad b_2 = c_2, \quad c_2 = d_2. \quad (8.11)$$

### 8.6.6 Intermezzo: Increasing Productivity

Uniformity in the formatting of the input makes it easier to relate the input and the output. In addition it helps spotting inconsistencies thereby reducing the possibility of errors in the input. Finally, it helps with debugging. For example, when you're creating complex output using the `align` environment it is a good idea to have one (or a few) number of aligned items per line. If you get an error in the input then you can easily comment out the equations, one at a time, until the error is gone, which should tell you there is something wrong in the vicinity of the last commented line in the input. If you define multiple equations on a single input line then finding the error may pose more problems. By treating LaTeX as a regular programming language you increase your productivity.



**Figure 8.1**

The `\shortintertext` command.

---

```
\begin{align*}
  x_0 &= 0\,,\, \\\
  x_1 &= 1\,,\, \\\
\shortintertext{and} & \text{and} \\
  x_2 &= 2\,,\, \\
\end{align*}
```

---


$$\begin{array}{lcl}
 x_0 = 0, & & \\
 x_1 = 1, & & \\
 & \text{and} & \\
 x_2 = 2. & &
 \end{array}$$


---

**Figure 8.2**

The `aligned` environment.

---

```
\begin{equation*}
  I = \left[
    \begin{aligned}
      1 &\& 0 &\& 0 \\\
      0 &\& 1 &\& 0 \\\
      0 &\& 0 &\& 1
    \end{aligned}
  \right],\, .
\end{equation*}
```

---


$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$


---

### 8.6.7 Interrupting a Display

The `amsmath` package also provides a command called `\intertext` for a short interjection of one or lines in the middle of a multi-line display. The command `\shortintertext`, which is provided by the `mathtools` package, has a similar purpose but it takes less space. Figure 8.1 shows how you use it. (The command `\intertext` works in a similar way.) Notice that the equation symbols of all resulting equations are properly aligned.

### 8.6.8 Low-level Alignment Building Blocks

All alignment environments we’ve seen so far operate at the “line” level. This means you cannot use them as parts of other constructs. The environments `aligned`, `alignedat`, and `gathered` allow you to align things at a lower level. Figure 8.2 provides an example of how to use the `aligned` environment. Notice that the environment does not do any numbering: the numbering is controlled by the enclosing environment. In the input in Figure 8.2 the commands `\left` and `\right` scale the left and right square brackets which act as delimiters of the construct which is built using `aligned`. The commands `\left` and `\right` are properly explained in Section 8.8.1. The example in Figure 8.2 should not be used as a general idiom for typesetting matrices. Better ways to typeset matrices are explained in Section 8.14. More information about the other low-level alignment commands may be found in the `amsmath` documentation [American Mathematical Society, 2002].

### 8.6.9 The eqnarray Environment

Standard  $\text{\LaTeX}$  also has an `eqnarray` environment. This environment is traditionally used for typesetting multiple equations with vertical alignment. The output which you get from this environment it is not always satisfactory.  $\text{\TeX}$ perts strongly recommend that you use the `amsmath` alignment primitives instead.

## 8.7 Text in Formulae

Every now and then you need plain text in mathematical formulae. The `amsmath` package provides a command `\text` which lets you do this. Using it, as is demonstrated by the following example, is easy.

<pre>\[ \text{final}       = \text{mark for assignments} \times       + 5 \times \text{mark for literature review} \]</pre>	$\text{\LaTeX}$ Input
---	-----------------------

This gives you:

$\text{final} = \text{marks for assignments} + 5 \times \text{mark for literature review}.$	$\text{\LaTeX}$ Output
---	------------------------

Inside the argument of the `\text` command you can safely switch to ordinary math mode and back. This also allows you to have a `\text` command inside the argument of a `\text` command. This makes writing `\text{f $\text{f $f$}$}` perfectly valid (but not particularly meaningful).

## 8.8 Delimiters

This section studies *delimiters*, which occur naturally in mathematical expressions. For example, the opening and closing parentheses act as delimiters of the start and end of the argument list of a function:  $f(a)$ ,  $g(x, y)$ , and so on. Likewise, the symbol  $| \cdot |$  is used as a left and right delimiter in the commonly used notation,  $|\cdot|$ , for absolute values. Despite the importance of delimiters,  $\text{\LaTeX}$  is not always unaware of their purpose and rôle in expressions. As a result it may sometimes use the wrong size and spacing in expressions with delimiters.

The remainder of this section helps you typeset your delimiters in the right size and with the correct spacing relative to the rest of your expressions. Section 8.8.1 starts by describing the commands `\left` and `\right`, which are used for scaling delimiters. This is followed by Section 8.8.2, which describes how to typeset bar-shaped delimiters depending on their context. Section 8.8.3 shows the proper commands for typesetting tuples. Section 8.8.4 does the same for floor and ceiling expressions. The section concludes with Section 8.8.5, which provides a list of frequently used variable-size delimiter commands.

### 8.8.1 Scaling Left and Right Delimiters

We've already seen the commands `\left` and `\right` as part of an example, but this section properly describes the purpose of these commands. The main purpose of the commands `\left` and `\right` is to typeset *variable-sized* delimiters in the proper size.

To understand why we sometimes need to scale delimiters, consider the (artificial)  $\text{\LaTeX}$  expression `$f( 2^{\{2^2\}}_{\{2_{22}\}} )$`. If we typeset it using  $\text{\LaTeX}$  this gives us  $f(2_{22}^{2^2})$ . The resulting output is not very pretty since the parentheses, which act as delimiters of the arguments of  $f(\cdot)$ , are too small.  $\text{\LaTeX}$  is simply not aware that the parentheses are serving as delimiters. To tell  $\text{\LaTeX}$  that the parentheses are left and right delimiters we make their purpose explicit by tagging them with `\left` and `\right`. This is done as follows: `$f\left( 2^{\{2^2\}}_{\{2_{22}\}} \right)$`, which gives us  $f\left(2_{22}^{2^2}\right)$ . You can use this technique for any kind of variable-sized delimiter symbol. Section 8.8.5 presents the different kinds of variable-sized delimiters.

You cannot use `\left` without `\right` and vice versa, which sometimes poses a problem. For example, how to typeset the following?

$$n! = \begin{cases} 1 & \text{if } n \leq 1, \\ n \times (n-1)! & \text{otherwise.} \end{cases}$$

$\text{\LaTeX}$  Output

The following is the solution. In the solution we use a `\right.` which balances the `\left-\right` pair and produces nothing. The construct `\left.` may be used similarly.

```
\[ n! =
\left\{
\begin{aligned}
& 1 && \&\& \text{if } \$n \leq 1\$ \\
& n \times (n-1)! && \&\& \text{otherwise}
\end{aligned}
\right. \]
```

$\text{\LaTeX}$  Input

Notice that the `\{` in `\left\{` in the input is not the left brace for starting a group, but the command for typesetting the left brace. The `cases` environment provides an easier way to define case-based definitions. The environment is explained in Section 8.17, which also discusses other solutions to case-based definitions.

Unfortunately you cannot have newlines in combination with `\left` and `\right`. The solution is to use the `\vphantom` command.

```
\begin{align*}
f &= g\left(3^{\{3^{\{3\}}\}} + \dots\right. \\
&\quad \&\& \left.+ 3 \vphantom{3^{\{3^{\{3\}}\}} + \dots}\right) \\
\end{align*}
```

$\text{\LaTeX}$  Input

If you use this you should get the following:

L<sup>A</sup>T<sub>E</sub>X Output

$$f = g \left( 3^{3^3} + \dots + 3 \right).$$

### 8.8.2 Bars

$\mathcal{A}\mathcal{M}\mathcal{S}$ -L<sup>A</sup>T<sub>E</sub>X provides several commands for typesetting vertical bars ( $|$ ). The reason for having several commands is that L<sup>A</sup>T<sub>E</sub>X's command `\vert` sometimes acts as a left, sometimes acts as a right delimiter, and sometimes acts as a different kind of delimiter. Depending on the rôle of the delimiter symbol it has to be treated differently, which is why  $\mathcal{A}\mathcal{M}\mathcal{S}$ -L<sup>A</sup>T<sub>E</sub>X provides dedicated commands which make the rôle of the delimiters explicit.

In the remainder of this section we shall first study the standard command `\vert` and then the special-purpose commands which are provided by  $\mathcal{A}\mathcal{M}\mathcal{S}$ -L<sup>A</sup>T<sub>E</sub>X. The following demonstrates how to typeset the vertical bar which occurs in “guarded” sets.<sup>1</sup>

The even digits are

L<sup>A</sup>T<sub>E</sub>X Input

```
\{\, 2 i \, \vert\, 0 \leq i \leq 4 \, \}\,
```

The following is the resulting output.

L<sup>A</sup>T<sub>E</sub>X Output

The even digits are  $\{2i \mid 0 \leq i \leq 4\}$ .

Using the thin spaces before and after the vertical bar is slightly better than omitting them. It may be argued that the use of a colon in guarded set notations is better than the use of a bar. For example,  $\{|i| \mid -10 \leq i \leq 9\}$  is much more difficult to parse than  $\{|i| : -10 \leq i \leq 9\}$ .

There are two more command-pairs for typesetting variable-size bars.

```
\left\lvert x \right\rvert
```

These commands are for absolute-values and similar:  $|x|$ .

□

```
\left\lVert x \right\rVert
```

These commands are for norms:  $\|x\|$ .

□

The `\rvert` command is also used for the “evaluation at” notation.

```
\left[ f(x) \right]_{x=0} = 0,
```

L<sup>A</sup>T<sub>E</sub>X Input

The following is the resulting output. Notice that the ‘`\left.`’ balances the `\left-``\right` pair.

L<sup>A</sup>T<sub>E</sub>X Output

$$f(x)|_{x=0} = 0.$$

### 8.8.3 Tuples

A common error in computer science and mathematical papers is to use ‘ $\langle 1, 2, 3 \rangle$ ’ for the tuple/sequence consisting of 1, then 2, and then 3.

<sup>1</sup>The adjective ‘guarded’ for sets is inspired by guarded lists in the Haskell programming language [Peyton Jones and Hughes, 1999].

This kind of  $\text{\LaTeX}$  input gives you ‘ $\langle 1, 2, 3 \rangle$ ’, which looks so bad that some authors have complained about this (see for example [Aslaksen, 1993]).  $\text{\LaTeX}$  has a special  $\text{\textbackslash langle}$  and  $\text{\textbackslash rangle}$  for tuples. If you use them for tuples then the result will look much more aesthetically pleasing. The following provides an example.

Let  $F(z)$  be the ordinary generating function of the sequence  $\langle t_0, t_1, \dots \rangle$ , then  $zF(z)$  is the ordinary generating function of the sequence  $\langle 0, t_0, t_1, \dots \rangle$ .

The following is the resulting output.

Let  $F(z)$  be the ordinary generating function of the sequence  $\langle t_0, t_1, \dots \rangle$ , then  $zF(z)$  is the ordinary generating function of the sequence  $\langle 0, t_0, t_1, \dots \rangle$ .

#### 8.8.4 Floors and Ceilings

The commands  $\text{\textbackslash lfloor}$  and  $\text{\textbackslash rfloor}$  are for typesetting “floor” expressions which are used to express rounding down. The two related commands  $\text{\textbackslash lceil}$  and  $\text{\textbackslash rceil}$  are for typesetting “ceiling” expressions. They are for rounding up. The following provides an example.

Let  $x$  be any real. By definition 
$$\begin{aligned} i &\leq \lfloor x \rfloor \leq x \leq \lceil x \rceil \leq I \\ &\text{for all integers } i \text{ and } I \text{ such that } i \leq x \leq I. \end{aligned}$$

The output looks like this.

Let  $x$  be any real. By definition 
$$i \leq \lfloor x \rfloor \leq x \leq \lceil x \rceil \leq I$$
 for all integers  $i$  and  $I$  such that  $i \leq x \leq I$ .

#### 8.8.5 Delimiter Commands

This section presents some more commands for variable-sized delimiters. Table 8.3 lists the commands. This table is based on [Pakin, 2005, Tables 74 and 76].

### 8.9 Fractions

This section is about typesetting fractions in math mode. Ordinary fractions are typeset using the command  $\text{\textbackslash frac}$ . To get the fraction  $\frac{\langle \text{num} \rangle}{\langle \text{den} \rangle}$  you use  $\text{\textbackslash frac}\{\langle \text{num} \rangle\}\{\langle \text{den} \rangle\}$ . Notice that fractions in the running text may disturb the flow of reading because they may increase the interline spacing. When using the  $\text{\textbackslash frac}$  command in inline math mode you should ensure that the resulting interline spacing is acceptable. If it is not

Table 8.3

This table lists variable-size delimiters and the commands to typeset them. All delimiters are typeset in inline math mode. The delimiters listed under the heading ‘Standard’ are standard L<sup>A</sup>T<sub>E</sub>X-provided commands. The delimiters listed under the heading ‘amsmath’ are provided by *AMS-L<sup>A</sup>T<sub>E</sub>X*. All commands can be used with or without `\left` and `\right`.

Standard		
<code>{ \{</code>	<code>} \}</code>	<code>\langle \langle</code>
<code>[ \lceil</code>	<code>⌊ \lfloor</code>	<code>\rangle \rangle</code>
<code>] \rceil</code>	<code>⌋ \rfloor</code>	<code>\uparrow \uparrow</code>
<code>\Downarrow</code>	<code>\updownarrow</code>	<code>\downarrow \downarrow</code>
<code>\Uparrow</code>	<code>\Updownarrow</code>	<code>( (</code>
<code>[ [</code>	<code>   </code>	<code>) )</code>
<code>] ]</code>	<code>   \lvert</code>	<code>/ /</code>
<code>\ \backslash</code>		
amsmath		
<code>\lvert</code>	<code>\rvert</code>	
<code>\lVert</code>	<code>\rVert</code>	

then perhaps it is possible to eliminate the division from your fractions. For example, a simple equation of the form  $x = \frac{1}{3}y$  is equivalent to  $3x = y$ . If you can’t eliminate division then perhaps you can turn the `\frac` construct into a simple `\langle num \rangle / \langle den \rangle` construct. Alternatively, you can typeset the fraction in a display.

The *amsmath* package provides a specialised command `\cfrac` for typesetting continued fractions. The command takes an optional argument for the placement of the numerator. The value of this optional argument may be either ‘l’ for left placement or ‘r’ for right placement: you may write `\cfrac[l]{\langle num \rangle}{\langle den \rangle}` or `\cfrac[r]{\langle num \rangle}{\langle den \rangle}`. The following provides an example of how to use the command. In the example, the command `\dotsb` is for ellipsis in combination with binary operators or relations.

```
\[ F = \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \dotsb}}} \]
```

$$F = \frac{1}{2 + \frac{1}{2 + \frac{1}{2 + \dots}}}$$

8.10 Sums, Products, and Friends

This section describes how to typeset sums, product, integrals, and related constructs. Section 8.10.1 explains the basic typesetting commands. Section 8.10.2 explains how to get more control over the typesetting of lower and upper limits of delimited sums, products, and so on. This section concludes with Section 8.10.3 which explains how to create multi-line upper and lower limits.

### 8.10.1 Basic Typesetting Commands

This section explains how to do basic typesetting of sums, products, and related constructs. To get started we shall study typesetting sums.

The *undelimited* sum symbol,  $\sum$ , may be typeset in *math mode* using the command `\sum`. It cannot be used in text mode.

In the *delimited* version the summands are parameterised by an index which ranges from a lower to an upper limit. The subscript (`_`) and superscript (`^`) operators are used to define the lower and upper limits of these delimited sums. So `\sum_{i=0}^n f(i)` defines the delimited sum with summand  $f(i)$  and lower and upper limits for the index variable  $i$  which are given by 0 and  $n$  respectively. The notation  $\sum_{i=0}^n f(i)$  is a shorthand for  $f(0) + f(1) + f(2) + \cdots + f(n)$ .

In the *generalised* summation notation [Graham, Knuth and Patashnik, 1989, Page 22] the range of the index variable is defined as a condition which is defined in the same position as the lower limit. Examples of this form are  $\sum_{0 \leq k < n} 2^{-k}$  and

$$\sum_{0 \leq k \leq n, \text{ odd } k} 2^k.$$

If you study how the last two sums in the previous paragraph are typeset then you may notice that their limits are typeset in different positions (relative to the  $\sum$  symbol). This is not a coincidence. Indeed, in a display the limits usually appear below and above the summation symbol. However, in inline math mode they are positioned to the lower and upper right of the summation symbol. For inline math mode this avoids annoying discrepancies in interline spacing.

The following provides one more example of how to typeset delimited sums.

L<sup>A</sup>T<sub>E</sub>X Input

It is well known that

$$\sum_{n=0}^{\infty} 2^{-n} = 2.$$

However, it is less well known that the Trie Sum,  $S_N$ , satisfies the following property as  $N \rightarrow \infty$ :

[Sedgewick:Flajolet:96]:

$$S_N = \sum_{n=0}^{\infty} \left( 1 - \left( 1 - 2^{-n} \right)^N \right) \\ = \lg N + O\left( \log \log N \right).$$

\]

The following is the resulting output. Again notice that the limits are different for ordinary and displayed math.

Table 8.4

This table lists variable-sized symbols and the commands to typeset them. All commands are typeset in ordinary math mode. The commands in the first four rows of the table are standard L<sup>A</sup>T<sub>E</sub>X commands. The commands in the last row of the table are provided by the `amsmath` package.

Standard		
$\sum$ <code>\sum</code>	$\int$ <code>\int</code>	$\bigcap$ <code>\bigcap</code>
$\prod$ <code>\prod</code>	$\oint$ <code>\oint</code>	$\bigcup$ <code>\bigcup</code>
$\oplus$ <code>\bigoplus</code>	$\sqcup$ <code>\bigsqcup</code>	$\bigwedge$ <code>\bigwedge</code>
$\otimes$ <code>\bigotimes</code>	$\coprod$ <code>\coprod</code>	$\bigvee$ <code>\bigvee</code>
$\odot$ <code>\bigodot</code>	$\biguplus$ <code>\biguplus</code>	
A <sub>M</sub> S- <sup>E</sup> T <sub>E</sub> X		
$\iint$ <code>\iint</code>	$\iiint$ <code>\iiint</code>	$\iiiiiint$ <code>\iiiiiint</code>
$\int \cdots \int$ <code>\idotsint</code>		

It is well known that  $\sum_{n=0}^{\infty} 2^{-n} = 2$ . However, it is less well known that the Trie Sum,  $S_N$ , satisfies the following property as  $N \rightarrow \infty$  [Sedgewick and Flajolet, 1996, Theorem 4.10]:

$$S_N = \sum_{n=0}^{\infty} \left(1 - (1 - 2^{-n})^N\right) = \lg N + O(\log \log N).$$

L<sup>A</sup>T<sub>E</sub>X Output

The  $\sum$  symbol is an example of a *variable-sized* symbol [Lamport, 1994, Page 44]. Table 8.4 lists variable-sized symbols and the commands to typeset them. All the commands in the table are used in exactly the same way as you use the command `\sum`. The top of the table is based on [Lamport, 1994, Table 3.8]. The commands in the top of the table are standard L<sup>A</sup>T<sub>E</sub>X commands. The commands in the last two rows are provided by the `amsmath` package.

8.10.2 Overriding the Basic Typesetting Style

Sometimes it is useful to change the way a variable-sized symbol is typeset. For example, a delimited sum which occurs in the numerator of a displayed fraction may look better if its limits are positioned to the lower and upper right of the  $\sum$  symbol. The commands `\textstyle` and `\displaystyle` allow you to override the default way the variable-sized symbols are typeset. The following provides an example of how to use the `\textstyle` command. The command `\displaystyle` is used similarly.

```
\[ \frac{\textstyle\sum_{n=0}^{\infty} 2^{-n}}{2}
```

= 1\,

```
\]
```

L<sup>A</sup>T<sub>E</sub>X Input

The following is the resulting output.

$$\frac{\sum_{n=0}^{\infty} 2^{-n}}{2} = 1.$$

L<sup>A</sup>T<sub>E</sub>X Output



### 8.10.3 Multi-line Limits

The command `\substack` lets you construct multi-line limits. The following demonstrates how it may be used in combination with the command `\sum`.

```
\[ \sum_{\substack{\text{$i$ odd}\\0 \leq i \leq n}} \binom{n}{i} \\
= 2^n - \\
\sum_{\substack{\text{$i$ even}\\0 \leq i \leq n}} \binom{n}{i}, . \\
\]
```

The following is the resulting output. As you may see from the input and the output the `\\` command is used to specify a newline within the stack. All layers in the stack are centred.

$$\sum_{\substack{i \text{ odd} \\ 0 \leq i \leq n}} \binom{n}{i} = 2^n - \sum_{\substack{i \text{ even} \\ 0 \leq i \leq n}} \binom{n}{i}.$$

The subarray environment gives you more control than `\substack`. The environment takes one more parameter which specifies the alignment of the layers in the stack. The extra parameter can be 'l' for alignment to the left or 'c' for alignment to the centre. The following demonstrates how the environment may be used to force different alignments of the two layers in the lower limits of the sums.

```
\[ \sum_{\begin{substack}{l} \\ i \text{ \texttt{\_odd}} \\ 0 \leq i \leq n \end{substack}} \binom{n}{i} \\
= 2^n - \\
\sum_{\begin{substack}{c} \\ i \text{ \texttt{\_even}} \\ 0 \leq i \leq n \end{substack}} \binom{n}{i}, . \]
```

The following is the resulting output. It may not be clear from the example but the spaces in the output which are generated before the 'odd' and 'even' are the result of the spaces which are part of the `\text` commands. They are typeset as visible spaces (`\_`) in the example. They are *not* caused by the spaces before the `\text` commands.

$$\sum_{\substack{i \text{ odd} \\ 0 \leq i \leq n}} \binom{n}{i} = 2^n - \sum_{\substack{i \text{ even} \\ 0 \leq i \leq n}} \binom{n}{i}.$$

Table 8.5  
Log-like functions.

<code>arccos</code>	<code>\arccos</code>	<code>dim</code>	<code>\dim</code>	<code>log</code>	<code>\log</code>
<code>arcsin</code>	<code>\arcsin</code>	<code>exp</code>	<code>\exp</code>	<code>max</code>	<code>\max</code>
<code>arctan</code>	<code>\arctan</code>	<code>gcd</code>	<code>\gcd</code>	<code>min</code>	<code>\min</code>
<code>arg</code>	<code>\arg</code>	<code>hom</code>	<code>\hom</code>	<code>Pr</code>	<code>\Pr</code>
<code>cos</code>	<code>\cos</code>	<code>inf</code>	<code>\inf</code>	<code>sec</code>	<code>\sec</code>
<code>cosh</code>	<code>\cosh</code>	<code>ker</code>	<code>\ker</code>	<code>sin</code>	<code>\sin</code>
<code>cot</code>	<code>\cot</code>	<code>lg</code>	<code>\lg</code>	<code>sinh</code>	<code>\sinh</code>
<code>coth</code>	<code>\coth</code>	<code>lim</code>	<code>\lim</code>	<code>sup</code>	<code>\sup</code>
<code>csc</code>	<code>\csc</code>	<code>lim inf</code>	<code>\liminf</code>	<code>tan</code>	<code>\tan</code>
<code>deg</code>	<code>\deg</code>	<code>lim sup</code>	<code>\limsup</code>	<code>tanh</code>	<code>\tanh</code>
<code>det</code>	<code>\det</code>	<code>ln</code>	<code>\ln</code>		

Figure 8.3  
Specifying the ‘limit’ argu-  
ment of existing log-like func-  
tions.



## 8.11 Functions and Operators

ℒ<sub>TEX</sub> comes with a wide range of commands for typesetting functions and operators. However, every  $\text{\TeX}$ nician some day has to face the problem of running out of symbols. Fortunately, the `amsmath` package provides a high-level command which lets you define your own commands for operators. The resulting operator symbol names are typeset properly and in a consistent style. This command gives you full control over the positioning of subscripts and superscripts in “limit” positions.

However, typesetting is only one part of the story. The `cool` package addresses the problem of capturing and dealing with the content of the mathematics.

The remainder of this section is as follows. Section 8.11.1 describes existing commands functions and operators. Section 8.11.2 describes the  $\mathcal{A}\mathcal{M}\mathcal{S}$ -ℒ<sub>TEX</sub>-provided command for defining your own function and operator symbols. Section 8.11.3 describes the `cool` package.

### 8.11.1 Existing Operators

The default type style for typesetting “log-like” function is math-roman (`\mathrm`). Table 8.5 lists ℒ<sub>TEX</sub>’s built-in log-like functions.

Some operators take subscripts and/or superscripts. They work as usual: the subscripts are specified with the subscript operator (`_`) and the superscripts with the superscript operator (`^`). Figure 8.3 demonstrates how to get the limit of the `\lim` command in the subscript position. Note that Figure 8.3 also works if we omit the braces which turn the second argument of the superscript operator into a group. Arguably, however, adding the braces makes the second argument stand out a bit.

The `mod` symbol is also overloaded. It requires different spacing de-

pending on the context. The `amsmath` package provides four commands to resolve this problem. The names of the commands are `\bmod`, `\mod`, `\pmod`, and `\pod`. They are used as follows.

`\bmod`

This is for binary modular division: ‘ $\gcd(5, 3) = \gcd(3, 5 \bmod 3)$ ’, which gives you ‘ $\gcd(5, 3) = \gcd(3, 5 \bmod 3)$ ’.  $\square$

`\mod`

This is for modular equivalence: ‘ $2 \equiv 5 \bmod 3$ ’, which gives you ‘ $2 \equiv 5 \bmod 3$ ’. Notice the difference in spacing compared to the spacing you get with the command `\bmod`. Here the operator symbol, `mod`, is further to the right of its first argument.  $\square$

`\pmod`

This is for parenthesised modular equivalence: ‘ $2 \equiv 5 \pmod 3$ ’, which gives you ‘ $2 \equiv 5 \pmod 3$ ’.  $\square$

`\pod`

This is for parenthesised modular equivalence without mod symbol: ‘ $2 \equiv 5 \pod 3$ ’, which gives you ‘ $2 \equiv 5 (3)$ ’.  $\square$

### 8.11.2 Declaring New Operators

$\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  provides the `\DeclareMathOperator` command for defining new operator names. The command can only be used in the preamble.

```
\DeclareMathOperator{<command>}{<sym>}
```

*LaTeX Usage*

This defines a new command, `<command>`, which is typeset as `<sym>`. The `<command>` should start with a backslash (`\`). The resulting symbol is typeset in a uniform style and with the proper spacing. The following is an example.

```
\documentclass{article}
\DeclareMathOperator{\bop}{binop}
\begin{document}
... Note that $1 \mathrm{binop}
2 = 3$ does not look pretty.
However, $1 \bop 2 = 3$ looks good.
```

*LaTeX Input*

It will give you the following output.

```
... Note that 1binop2 = 3 does not look pretty. However,
1 binop 2 = 3 looks good.
```

*LaTeX Output*

Notice that the appearance of both operator symbols is the same. However, the spacing for the first operator symbol is dreadful since  $\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  does not recognise it as an operator.

$\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  also provides a `\DeclareMathOperator*` command, which is for defining operator symbols that require subscripts and superscripts in “limit” positions. It can only be used in the preamble. The following is an example.

```

\documentclass{article}
\DeclareMathOperator*\Lim{\Lim}
\begin{document}
... $\Lim_{x \to 0} \frac{x^2}{x} = 0$. ...

```

It will give you the following output.

```

...  $\text{Lim}_{x \rightarrow 0} \frac{x^2}{x} = 0.$  ...

```

### 8.11.3 Managing Content with the cool Package

The cool package addresses the problem of capturing content.

- Provides *very* comprehensive list of commands for consistent typesetting of mathematical functions and constants.
- Provides commands for easy typesetting complex matrices.
- Provides commands which affect the way symbols and expressions are typeset. This affects:
  - How inverse trigonometric functions are typeset:  $\arcsin x$  versus  $\sin^{-1} x$ .
  - How derivatives are typeset:  $\frac{d}{dx} f$  versus  $\frac{df}{dx}$ .
  - How integrals are typeset:  $\int f \, dx$  versus  $\int dx \, f$ .
  - How certain function and polynomial symbols are printed.

## 8.12 Integration and Differentiation

### 8.12.1 Integration

The command `\int` is for typesetting simple integrals. The following demonstrates how to typeset definite integrals. Notice the standard `\left. - \right\rvert` trick for ensuring that the right bar has the correct size. Also notice the negative thin space before the `dx` in the input. (A negative thin space is required for each “d” part, so you write `\mathrm{d}` `\mathrm{d} x` `\mathrm{d} y`, and so on.)

```

\[ \int_a^b 3x^2 \mathrm{d} x
= \left. x^3 \right\rvert_a^b
= b^3 - a^3, .
\]

```

Notice that we also could have written the more terse `\mathrm{d} x` for the `\mathrm{d} x`. However, arguably, adding the braces is clearer. The following is the resulting output.

```


$$\int_a^b 3x^2 \, dx = x^3 \Big|_a^b = b^3 - a^3.$$


```

The key to typesetting more exotic integrals are the the commands which are provided by the `amsmath` and the `esint` packages. Table 8.6

**Table 8.6**

This table lists integration signs and the commands to typeset them. All commands are provided by the `amsmath` package.

$\int$ <code>\int</code>	$\iint$ <code>\iint</code>	$\iiint$ <code>\iiint</code>
$\iiint$ <code>\iiint</code>	$\int \cdots \int$ <code>\int \cdots \int</code>	

lists the commands which are provided by `amsmath`.<sup>2</sup>

## 8.12.2 Differentiation

Expressions with differentiations are typeset using the `\frac` command. The expression  $\frac{du}{dx}$  may be obtained with `\frac{\mathrm{d} u}{\mathrm{d} x}`. More complex expressions work as expected, so  $\frac{d^2u}{dx^2}$  may be obtained with `\frac{\mathrm{d}^2 u}{\mathrm{d} x^2}`.

The symbol  $\partial$  is typeset with the command `\partial`. The following provides an example.

`\[ \frac{\partial u}{\partial t}`  
`= h^2`  
`\left( \frac{\partial^2 u}{\partial x^2}`  
`+ \frac{\partial^2 u}{\partial y^2}`  
`+ \frac{\partial^2 u}{\partial z^2}`  
`\right)\,, .`  
`\]`

*LaTeX Input*

The resulting output is as follows.

$$\frac{\partial u}{\partial t} = h^2 \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right).$$

*LaTeX Output*

Notice that the symmetry in the output can be mimicked in the input by “stacking” the commands that typeset the three expressions inside the parentheses. Using this formatting style, any error in the input should be relatively easy to detect.

## 8.13 Roots

Square roots and other roots are typeset with `\sqrt`. The command has an optional argument for the root indices. The following provides an example.

`... $\sqrt{2} \approx 1.414213562$ and`  
`$\sqrt[3]{27} = 3$.`

*LaTeX Input*

This gives you.

$$\dots \sqrt{2} \approx 1.414213562 \text{ and } \sqrt[3]{27} = 3.$$

*LaTeX Output*

Sometimes the placement of the root indices is not perfect:  $\sqrt[k]{k}$ .

<sup>2</sup>Due to some technical problem (too many math alphabets) the `esint`-provided symbols cannot be displayed.

The `amsmath` package provides two commands for fine-tuning the typesetting.

- `\leftroot{⟨number⟩}` moves the root index `⟨number⟩` “units” to the left. The unit is an arbitrary but convenient distance. Notice that `⟨number⟩` can be negative, in which case this results in moving the root index to the right.
- `\uproot{⟨number⟩}` moves the root index `⟨number⟩` units up.

Using these commands `\sqrt[\leftroot{-2}\uproot{2}\beta]{k}` gives us  $\sqrt[\beta]{k}$ .

## 8.14 Arrays and Matrices

Traditionally arrays were typeset using the `array` environment, which works similar as the `tabbing` environment. High-level commands for typesetting matrices are provided by the `amsmath` package. The following example uses  $\text{\LaTeX}$ 's built-in `array` environment to typeset a complex-ish construct.

```
\[ \left( \begin{array}{c}
  \left\lvert \begin{array}{lrc}
    x & y & z \\
    0 + 1 + 2 & \alpha + \beta + \gamma & a + b + c \\
    \end{array} \right\rvert \\
  A \\
  B
\end{array} \right) \]
```

*$\text{\LaTeX}$  Input*

The resulting output is as follows.

$$\left( \left| \begin{array}{lrc} x & y & z \\ 0 + 1 + 2 & \alpha + \beta + \gamma & a + b + c \\ A & & \\ B & & \end{array} \right| \right)$$

*$\text{\LaTeX}$  Output*

The `amsmath` package provides the following high-level environments for typesetting matrices.

**`pmatrix`** for matrices with `( )` delimiters.

**`bmatrix`** for matrices with `[ ]` delimiters.

**`Bmatrix`** for matrices with `{ }` delimiters.

**`vmatrix`** for matrices with `| |` delimiters.

**`Vmatrix`** for matrices with `|| ||` delimiters.

**`matrix`** for matrices without delimiters.

All these commands are designed for displayed math mode. These commands do not let you specify vertical alignment. By default there are up to ten columns, which are aligned to the centre.

$\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  also provides a `smallmatrix` environment for typesetting matrices in inline (ordinary) math mode. The `smallmatrix` environment does not typeset the delimiters. To typeset the delimiters you use the commands `\bigl` and `\bigr`, which are the equivalents of the commands `\left` and `\right` respectfully, so for square bracket delimiters you write: `\bigl[\begin{smallmatrix} ... \end{smallmatrix}\bigr]`.

The following is another example.

$\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  Input

```
... Using matrices the linear transformation
 $\langle x, y \rangle$ 
 $\mapsto$ 
 $\langle 2x + y, 3y \rangle$ 
is written as follows:
 $\bigl[\begin{smallmatrix} 2&1 \\ 0&3 \end{smallmatrix}\bigr]$ 
 $\bigr[\begin{smallmatrix} x & y \end{smallmatrix}\bigr]$ .
You probably knew this already. ...
```

The following is the corresponding output.

$\mathcal{E}\mathcal{T}\mathcal{E}\mathcal{X}$  Output

```
... Using matrices the linear transformation  $\langle x, y \rangle \mapsto$ 
 $\langle 2x + y, 3y \rangle$  is written as follows:  $\begin{bmatrix} 2 & 1 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$ . You prob-
ably knew this already. ...
```

## 8.15 Math Mode Accents, Hats, and Other Decorations

This section is about typesetting accents and other decorations in math mode. The commands in this section are all of the form `\command{argument}`. The majority of the commands add a fixed-size decoration to the `argument`. For example, `\hat{x}` and `\hat{xxx}` give you  $\hat{x}$  and  $\hat{xxx}$ . The remaining commands provide extensible decorations. For example, the commands `\overline{x}` and `\overline{xxx}` give you  $\overline{x}$  and  $\overline{xxx}$ . The result may not always be aesthetically pleasing. For example, `\widetilde{xxxxxx}` gives you  $\widetilde{xxxxxx}$ . Table 8.7 lists some commonly used commands.

## 8.16 Braces

A common chore is that of typesetting expressions with overbraces ( $\overbrace{\langle \text{expr} \rangle}$ ) or underbraces ( $\underbrace{\langle \text{expr} \rangle}$ ). The commands for creating such expressions are `\overbrace` and `\underbrace`. As should be clear from the

Table 8.7

Math mode accents, hats, and other decorations. The commands at the top are listed with single letter arguments. They are intended for “narrow” arguments such as letters, digits, and so on. The commands at the bottom produce extensible decorations. The commands `\ddot` and `\ddddot` are provided by `ams-math`.

Fixed-size Decorations	
$\dot{x}$ <code>\dot{x}</code>	$\acute{x}$ <code>\acute{x}</code>
$\ddot{x}$ <code>\ddot{x}</code>	$\grave{x}$ <code>\grave{x}</code>
$\dddot{x}$ <code>\dddot{x}</code>	$\hat{x}$ <code>\hat{x}</code>
$\ddddot{x}$ <code>\ddddot{x}</code>	$\tilde{x}$ <code>\tilde{x}</code>
$\mathring{x}$ <code>\mathring{x}</code>	$\bar{x}$ <code>\bar{x}</code>
$\check{x}$ <code>\check{x}</code>	$\vec{x}$ <code>\vec{x}</code>
$\breve{x}$ <code>\breve{x}</code>	
Extensible Decorations	
$\overleftarrow{e}$ <code>\overleftarrow{e}</code>	$\overline{e}$ <code>\overline{e}</code>
$\overrightarrow{e}$ <code>\overrightarrow{e}</code>	$\widetilde{e}$ <code>\widetilde{e}</code>
$\overleftrightarrow{e}$ <code>\overleftrightarrow{e}</code>	$\widehat{e}$ <code>\widehat{e}</code>
$\underleftarrow{e}$ <code>\underleftarrow{e}</code>	$\underline{e}$ <code>\underline{e}</code>
$\underrightarrow{e}$ <code>\underrightarrow{e}</code>	$\underline{\underline{e}}$ <code>\underline{\underline{e}}</code>
$\underrightarrow{e}$ <code>\underrightarrow{e}</code>	

disruption in inter-line spacing, the use of overbraces and underbraces should be restricted to displayed math mode.

Expressions with underbraces can be “decorated” with expressions under the brace. Likewise, expressions with overbraces may receive decorations over the brace. These more complicated expressions are constructed using subscript and superscript operators. The following demonstrates how to use the `\overbrace` and `\underbrace` commands.

<code>\overbrace{⟨under⟩}</code>	This command gives you $\overbrace{\langle \text{under} \rangle}$ .	<input type="checkbox"/>
<code>\overbrace{⟨under⟩}^{\langle over \rangle}</code>	This command results in $\overbrace{\langle \text{under} \rangle}^{\langle \text{over} \rangle}$ .	<input type="checkbox"/>
<code>\underbrace{⟨under⟩}</code>	This command gives $\underbrace{\langle \text{under} \rangle}$ .	<input type="checkbox"/>
<code>\underbrace{⟨over⟩}_{\langle under \rangle}</code>	This command results in $\underbrace{\langle \text{over} \rangle}_{\langle \text{under} \rangle}$ .	<input type="checkbox"/>

The decorated versions are usually needed to indicate numbers of subterms. The following is an example. (Notice the use of the `\text` command to temporarily switch to text mode inside math mode.)



```
\[
  \underbrace{1 \times x \times x \times \dots \times x}_{k \text{ times } x} = x^k.
\]
```

The following is the resulting output.

$$\underbrace{1 \times x \times x \times \dots \times x}_{k \text{ times } x} = x^k.$$

## 8.17 Case-based Definitions

Case-based definitions are very common in computer science. There are two common approaches and solutions: conditions and *Iversonians*. The following explains these approaches in further detail.

**Conditions** With this approach we have conditions for each different case. The following provides an example.

```
\[ n! = \begin{cases} 1 & \text{if } n = 0 \\ (n-1)! \times n & \text{if } n > 0 \end{cases}
\]
```

This gives you the following.

$$n! = \begin{cases} 1 & \text{if } n = 0; \\ (n-1)! \times n & \text{if } n > 0. \end{cases}$$

The disadvantage of this kind of definition is that it is not very suitable for ordinary math mode.

**Iversonians** Here we define a 1-ary “characteristic function” which returns 1 if its argument is true and returns 0 otherwise. In [Graham, Knuth and Patashnik, 1989] the authors propose the notation  $[cond]$ , which they call the *Iversonian* of *cond*. Iversonian is a tribute to Kenneth E. Iverson, the inventor of the computer language A Programming Language (APL), which has a similar construct. The expression evaluates to 1 if *cond* is true and 0 otherwise. The notation  $1_{\{cond\}}$  is another accepted notation, but it has the disadvantage that it has a subscript. The following is an example.

```
... We define
$n! = [n = 0] + (n-1)! \times n \times [n > 0]$. ...
```

This gives you:

```
... We define  $n! = [n = 0] + (n-1)! \times n \times [n > 0]$ . ...
```

## 8.18 Function Definitions

Function definitions usually come with a description of the *domain*, the *range*, and the “*computation rule*.” The following provides an example.

The successor function, L<sup>A</sup>T<sub>E</sub>X Input  
 $\$s \colon \mathbb{N} \rightarrow \mathbb{N}\$,$   
 is defined as follows:  
 $\backslash[$   
 $s(n) \mapsto n + 1\backslash,$   
 $\backslash]$

The following is the resulting output.

The successor function,  $s : \mathbb{N} \rightarrow \mathbb{N}$ , is defined as follows: L<sup>A</sup>T<sub>E</sub>X Output  

$$s(n) \mapsto n + 1.$$

Note that the commands `\to` and `\mapsto` result in different arrows. Also note that using a colon (`:`) instead of the command `\colon` does not result in the correct spacing: it gives ‘ $s : \mathbb{N} \rightarrow \mathbb{N}$ ’. The `\mathbb` command typesets its argument in *blackboard* font. This is useful for typesetting the symbols  $\mathbb{N}$ ,  $\mathbb{Z}$ ,  $\mathbb{Q}$ ,  $\mathbb{R}$ ,  $\mathbb{C}$ , and other related symbols. The command can only be used in math mode.

## 8.19 Theorems

The package `amsthm` makes writing theorems, lemmas, and friends easy. The package ensures consistent numbering and appearance of theorem-like environments. The package provides:

- A proof environment;
- Styles for theorem-like environments;
- Commands for defining new theorem-like styles; and
- Commands for defining new theorem-like environments.

Section 8.19.1 is an introduction to the building blocks of theorems. This is followed by Section 8.19.2, which presents the default styles for theorem-like environments. Section 8.19.3 describes how to define theorem-like environments. Section 8.19.4 explains how to define new styles for theorem-like environments. Section 8.19.5 explains how to typeset proofs.

### 8.19.1 Ingredients of Theorems

The following is the typical output of a theorem-style environment.

**Theorem 2.1.3** (Fermat’s Last Theorem). *Let  $n > 2$  be any integer, then the equation  $a^n + b^n = c^n$  has no solutions in positive integers  $a$ ,  $b$ , and  $c$ .* L<sup>A</sup>T<sub>E</sub>X Output

The definition consists of several parts.

**Heading** The heading should describe the rôle of the environment. In this example the heading is ‘Theorem’. Usually, headings are Theorem, Lemma, Definition, and so on.

**Number** The number is optional and is used to refer to the environment in the running text. This is done using the usual `\label-\ref` mechanism. Numbers may depend on sectional units. If the number depends on sectional units then it is of the form  $\langle \text{unit} \rangle . \langle \text{number} \rangle$ , where  $\langle \text{unit} \rangle$  is the number of the current sectional unit, and  $\langle \text{number} \rangle$  is a number which is local within the sectional unit. If the number does not depend on the sectional unit then it is a plain number. In this example, the number of the environment is 2.1.3. This indicates that the number depends on the sectional unit 2.1 — probably Chapter 2.1 or Section 2.1 — and that within the unit this is the third instance.

**Body** The body of the environment is the text which conveys the message of the environment.

**Name** The name is optional. It serves two purposes. Most importantly, the name should capture the essence of the body. Next, it may be used to refer to the environment by name, as opposed by number (using `\ref`). In this example, the name of the environment is ‘Fermat’s Last Theorem’.

### 8.19.2 Theorem-like Styles

There are three existing styles for theorem-like environments: `plain`, `definition`, and `remark`. New styles may also be defined; this is explained in Section 8.19.4. The following explains the differences between the existing styles.

**plain** Usually associated with: Theorem, Lemma, Corollary, Proposition, Conjecture, Criterion, and Algorithm. The following demonstrates the appearance of the `plain` style.

<p><b>Theorem 1.1</b> (Fermat’s Last Theorem). <i>Let <math>n &gt; 2</math> be any integer, then the equation <math>a^n + b^n = c^n</math> has no solutions in positive integers <math>a</math>, <math>b</math>, and <math>c</math>.</i></p>	<p>LaTeX Output</p>
--	---------------------

**definition** Usually associated with: Definition, Condition, Problem, and Example. The following demonstrates the appearance of the `definition` style.

<p><b>Definition 1.2</b> (Ceiling). The <i>ceiling</i> of real number, <math>r</math>, is the smallest integer, <math>i</math>, such that <math>r \leq i</math>.</p>	<p>LaTeX Output</p>
--	---------------------

**remark** Usually associated with: Remark, Note, Notation, Claim, Summary, Acknowledgement, Case, and Conclusion. The following demonstrates the appearance of the `remark` style.

**Tip 1.3 (Tip).** Don't do this at home.

LaTeX Output

Numbering may or may not depend on the sectional unit. The following explains the differences.

**Independent numbering** Here the numbers are integers. So if theorems are numbered continuously you may have Theorem 1, Theorem 2, Theorem 3, and so on.

**Dependent numbering** Here the numbers are of the form  $\langle \text{unit label} \rangle . \langle \text{local} \rangle$ , where  $\langle \text{unit label} \rangle$  depends on the number/label of the sectional unit (chapter, section, ...), and  $\langle \text{local} \rangle$  is a local number. With numbering dependent on a section in a book you may have Theorem 1.1.1, Theorem 1.1.2, Theorem 2.3.1, and so on.

Different environments may be numbered differently.

- Different environments may share the same number sequence. If this is the case you may get Theorem 1, Lemma 2, Theorem 3, and so on, but not Theorem 2.
- Environments may have their own *independent* number sequence. If this is the case you could get Theorem 1, Lemma 1, Theorem 2, and so on.

### 8.19.3 Defining Theorem-like Environments

Defining new theorem-like environment styles is done in two stages. *First* you set the current style, *next* you define the environments. The environments will all be typeset in the style which was current at the time of definition of the environments.

**Step 1: Defining the current style** Defining the current style is done with the `\theoremstyle` command. The command takes the label of the style as its argument. Initially, the current style is `plain`.

**Step 2: Defining the environments** Defining the environments is done with the `\newtheorem` command. Environments defined by `\newtheorem` will be typeset according to the style which was current at the time of definition. The numbering and headings of the environments are determined by the command `\newtheorem`, which takes an optional argument which may appear in different positions.

The remainder of this section explains the `\newtheorem` command. We shall first study how to use the command without the optional argument. Next we shall study how to use it with the optional argument. This section closes with an example.

*Without* the optional argument you define environments using `\newtheorem{env}{heading}`. This defines a new environment `env` with heading `heading`. The environment is started with a new numbering

**Figure 8.4**  
Using the `amsthm` package.

---

```

\usepackage{amsmath}
\usepackage{amsthm}

% Current environment style is plain.
%% Define environment thm for theorems.
\newtheorem{thm}{Theorem}
%% Define environment lemma for lemmas.
%% Share numbering of with thm environment.
\newtheorem{lemma}[thm]{Lemma}

% Set environment style to definition.
\theoremstyle{definition}
%% Define environment def for definitions.
%% Share numbering with thm environment.
\newtheorem{def}[thm]{Definition}

```

---

sequence. For example, to define a new environment called `thm` for theorems with a new numbering sequence you would use `\newtheorem{thm}{Theorem}`.

*With* the optional argument, the optional argument may be used in different positions. It may be used as the *second* argument and as the *last* argument. The following explains the differences.

- If the optional argument is used as the *second* argument of the `\newtheorem` command, then you define the environment using `\newtheorem{<env>}[<old>]{<heading>}`. This defines a new environment `<env>` with heading text `<heading>`. The environment does not start with a new numbering sequence. Instead, the environment shares its numbering with the existing theorem-style environment `<old>`.
- If the optional argument is used as the *last* argument, you define the environment using `\newtheorem{<env>}{<heading>}[<unit>]`. This defines a new environment `<env>` with heading `<heading>`. The argument `<unit>` should be the name of a sectional unit, for example, `chapter`, `section`, .... This defines an environment called `<env>` with heading `<heading>` and a new numbering sequence which depends on the sectional unit `<unit>`.

Figure 8.4 provides an example of how the `amsthm` package may be used to define three theorem-like environments called `thm`, `lem`, and `def` with headings `Theorem`, `Lemma`, and `Definition`. The first two environments are typeset in the style `plain`. The last environment is typeset in the style `definition`. The numbering of the environments does not depend on sectional units and is shared.

#### 8.19.4 Defining Theorem-like Styles

The command `\newtheoremstyle` is for defining new `amsmath` theorem-

**Table 8.8**  
Math mode dot-like symbols.

<code>\ldotp</code>	<code>\ldots</code>	<code>\cdotp</code>
<code>\cdots</code>	<code>\colon</code>	<code>\vdots</code>
<code>\ddots</code>		

like environment styles. This command gives you ultimate control over fine typesetting of the environments. Usually the predefined styles `plain`, `definition`, and `remark` suffice. Exact information about the command `\newtheoremstyle` may be found in the `amsthm` documentation [American Mathematical Society, 2004].

8.19.5 Proofs

Writing proofs is done with the `proof` environment. The environment takes an optional argument for a title of the proof. The environment makes sure that it completes the proof by putting a square ( $\square$ ) at the end of the proof. This makes it easy to recognise the end of the proof. Unfortunately, the automatic mechanism for putting the square at the end of the proof doesn't work well if the proof ends in a displayed formula. To overcome this problem, there is also a command called `\qedhere` for putting the square at the end of the last displayed formula. The following provides an example.

`\begin{proof}[Technical Challenge]`  
To prove that  $3^2 + 4^2 = 5^2$ , we note that  
`\[ 3^2 + 4^2 = 9 + 4^2 = 9 + 16 = 25 = 5^2 \]`, `\qedhere`  
`\]`  
`\end{proof}`

*Technical Challenge.* To prove that  $3^2 + 4^2 = 5^2$ , we note  
that  
$$3^2 + 4^2 = 9 + 4^2 = 9 + 16 = 25 = 5^2.$$
  $\square$

8.20 Mathematical Punctuation

$\TeX$  provides several commands for typesetting dot-like symbols. Table 8.8 lists  $\TeX$ 's built-in commands. Unfortunately, it is not quite clear how these commands should be used. The following provides some guidelines about how these symbols should be used.

<code>\ldotp</code>	Used for the definition of <code>\ldots</code> [Knuth, 1990, Page 438].	$\square$
<code>\ldots</code>	Low dots. Used between commas, and when things are juxtaposed with no signs between them at all [Knuth, 1990, Page 172]. For example, <code>\f(x_1, \ldots, x_n)</code> gives you $f(x_1, \dots, x_n)$ and <code>n(n-1)\ldots(1)</code> gives you $n(n-1)\dots(1)$ .	$\square$

**Figure 8.5**  
Using the mathematical punctuation commands.

<code>\ldots</code> Then we have series	...Then we have series $A_1, A_2, \dots$ ,
<code>\$A_1, A_2, \dotsc\$</code> ,	regional sum $A_1 + A_2 + \dots$ , or
<code>\$A_1 + A_2 + \dotsb\$</code> ,	thogonal product $A_1 A_2 \dots$ , and
<code>\$A_1 A_2 \dotsm\$</code> ,	infinite integral
<code>\$A_1 \int_{A_1} \int_{A_2} \dots\$</code>	
<code>\int_{A_1} \int_{A_2} \dots</code>	
<code>\int_{A_1} \int_{A_2} \dots</code>	
<code>\dotsi \backslash, . \backslash</code>	

<code>\cdotp</code>	Used for the definition of <code>\cdots</code> [Knuth, 1990, Page 438].	<input type="checkbox"/>
<code>\cdots</code>	Centred dots. Used between $+$ and $-$ and $\times$ signs, between $=$ signs and other binary relational operator signs [Knuth, 1990, Page 172]. For example, <code>\$x_{\{1\}}+\cdots+x_{\{n\}}\$</code> gives you $x_1 + \dots + x_n$ .	<input type="checkbox"/>
<code>\colon</code>	Used as punctuation mark [Knuth, 1990, Page 134]: <code>\$f \colon A \to B\$</code> .	<input type="checkbox"/>
<code>\ddots</code>	Used in arrays and matrices.	<input type="checkbox"/>
<code>\vdots</code>	Used in arrays and matrices.	<input type="checkbox"/>
	Notice that the command <code>\cdot</code> also produces a dot. However, this is not used for punctuation. It is generally used in expressions like $(x_1, \dots, x_n) \cdot (y_1, \dots, y_n)$ .	
	Many symbols occurring in mathematical formulae require different spacing depending on their context. The commands which reproduce these symbols are context-unaware. The <code>amsmath</code> package provides several commands to overcome this problem. The following commands are for typesetting dots and sequences of dots.	
<code>\dotsc</code>	For dots in combination with commas.	<input type="checkbox"/>
<code>\dotsb</code>	For dots in combination with binary operators/relations.	<input type="checkbox"/>
<code>\dotsm</code>	For multiplication dots.	<input type="checkbox"/>
<code>\dotsi</code>	For dots with integrals.	<input type="checkbox"/>
<code>\dotso</code>	For other dots.	<input type="checkbox"/>

Figure 8.5 demonstrates the effect of these commands. This figure is based on the `amsmath` documentation [American Mathematical Society, 2002].

## 8.21 Spacing and Linebreaks

This section provides some information and guidelines related to spacing and linebreaking in math mode. The majority of this section is based on [Knuth, 1990, Chapter 18].

### 8.21.1 Line Breaks

TeX may break lines after commas in text mode but it doesn't lines after commas in math mode. This makes sense since you don't want to see a break after the comma in ' $f(a, b)$ '. Make sure you keep the commas which are part of formulae inside the dollar expressions in ordinary math mode. The remaining commas should be kept outside. The following is correct.

for  $x = f(a, b)$ ,  $f(b, c)$ , or  $f(b, c)$ . L<sup>A</sup>T<sub>E</sub>X Usage

However, the following is not correct.

for  $x = f(a, b)$ ,  $f(b, c)$ , or  $f(b, c)$ . Don't Try this at Home

In displayed math mode the TeXpert is ultimately responsible for linebreaks and inserting whitespace. This is especially true in environments with vertical alignment. The following are a few guidelines.

- Always insert a thin space ( $\,$ ) before punctuation symbols at the end of the lines.
- In sums or differences linebreaks should be inserted *before* the plus or minus operator. On the next line you should insert a *qqquad* after the alignment position. Here a qqquad is equivalent to two quads. One quad is the equivalent of the width of the uppercase 'M'. If the continuation line is short you may even consider inserting several quads. You insert a single quad with the command `\quad`. A single qqquad is inserted with the command `\qqquad`.

```
\begin{align*}
f(x) &= a + b + c + d \\
      &\quad + e + f + g\,,
\end{align*}
```

L<sup>A</sup>T<sub>E</sub>X Usage

- Linebreaks in products should occur *after* the multiplication operator. The operator should be *repeated* on the next line.

```
\begin{align*}
f(x) &= a \times b \times c \times d \times \\
      &\quad \times e \times f \times g\,,
\end{align*}
```

L<sup>A</sup>T<sub>E</sub>X Usage

### 8.21.2 Conditions

In ordinary math mode, you should put an extra space for conditions following equations. This makes the conditions stand out a bit more.



The Fibonacci numbers satisfy

*LaTeX Usage*

```
$F_{n} = F_{n - 1} + F_{n - 2}$,
\ $n \geq 2$.
```

However, it is probably better to turn the previous example into a proper sentence as follows.

The Fibonacci numbers satisfy

*LaTeX Usage*

```
$F_{n} = F_{n - 1} + F_{n - 2}$,
for~$n \geq 2$.
```

If you need to add an additional condition to a formula in displayed math mode then the two should be separated with a single `qquad`.

```
\[ z^m G( z )
```

*LaTeX Usage*

```
= \sum_{n} g_{n - m} z^n \,,
\quad\text{integer $m \geq 0$} \,, .
```

```
\]
```

Alternatively, you can put the condition in parentheses. However, if you do this, you have to omit the comma before the condition.

```
\[ z^m G( z )
```

*LaTeX Usage*

```
= \sum_{n} g_{n - m} z^n
\quad\text{((integer $m \geq 0$))} \,, .
```

```
\]
```

### 8.21.3 Physical Units

Physical units should be typeset in roman (`\mathrm`). In expressions of the form `<number> <unit>`, you insert a thin space between the number and the unit: `<number> \, <unit>`. The following is a concrete example.

```
$g = 9.8 \, \mathrm{m} / \mathrm{s}^2$
```

*LaTeX Usage*

The `siunitx` package provides support for typesetting units. Using the package you write `\SI{9.8}{\metre\per\second\squared}`. This gives you  $9.8 \text{ m s}^{-2}$  as standard, or  $9.8 \text{ m/s}^2$  by setting `per=slash` with the `\sisetup` macro. More information about the `siunitx` package may be found in the package documentation [Wright, 2008].

### 8.21.4 Sets

Sets come in two flavours. On the one hand there are “ordinary” sets the definitions of which do not depend on conditions:  $\{1\}$ ,  $\{3, 5, 6\}$ , and so on. On the other hand there are “guarded set” whose definitions do depend on conditions:  $\{2n : n \in \mathbb{N}\}$  and so on.

For ordinary sets there is no need to add additional spacing after the opening brace and before the closing brace.

The natural numbers,  $\mathbb{N}$ , are defined

*LaTeX Usage*

```
$\mathbb{N} = \{ 0, 1, 2, \ldots \}$.
```

Table 8.9

This table demonstrates the effect of the positive and negative spacing commands. The first two columns list the positive spacing commands, the next two columns demonstrate the effect of the `\hphantom` command, and the last two columns list the negative spacing commands. It is assumed that all these commands are used in inline math mode. The horizontal space from the tip of the arrow pointing to the right to the tip of the arrow pointing to the left in the second column demonstrates the effect of the spacing. The spacing is negative if the arrow tips overlap horizontally.

Positive Spacing		\hphantom		Negative Spacing	
<code>\,</code>		<code>\hphantom{M}</code>		<code>\!</code>	
<code>\thinspace</code>		<code>M</code>		<code>\negthinspace</code>	
<code>\:</code>		<code>\hphantom{z^n}</code>		<code>\negmedspace</code>	
<code>\medspace</code>		<code>z^n</code>		<code>\negthickspace</code>	
<code>\;</code>					
<code>\thickspace</code>					
<code>\quad</code>					
<code>\qquad</code>					

For guarded sets you insert a thin space after the opening and before the closing brace. The use of a thin space before and after the colon is not recommended by [Knuth, 1990], but it may be argued that it makes the result easier to read.

The even numbers,  $E$ , are definedLaTeX Usage  
 $E = \left\{ \, , \, 2 \, n \, , \, : \, , \, n \, \in \, \mathbb{N} \, \, , \, \right\}.$

If you don't like the colon then you should write

The even numbers,  $E$ , are definedLaTeX Usage  
 $E = \left\{ \, , \, 2 \, n \, , \, \mid \, , \, n \, \in \, \mathbb{N} \, \, , \, \right\}.$

### 8.21.5 More Spacing Commands

Table 8.9 demonstrates the effect of the horizontal spacing commands. The command `\hphantom`, which is listed in Table 8.9, is related to the command `\phantom`. It results in a horizontal space which is equal to the width of its argument.

## 8.22 Changing the Style

The following six commands let you change the type style in math mode.

- `\mathit{italic + abc^2}`  
This typesets its argument in ‘math italics’: *italic + abc<sup>2</sup>*. ☐
- `\mathrm{roman + abc^2}`  
This typesets its argument in ‘math roman’: roman + abc<sup>2</sup>. ☐
- `\mathbf{bold + abc^2}`  
This typesets its argument in the default ‘math bold face’ font: **bold + abc<sup>2</sup>**.  
Notice that `\mathbf` may not always result in bold symbols. Although not ideal, the commands `\pmb` (poor man’s bold) and `\boldsymbol` may be useful in cases like this. ☐

**Table 8.10**

This table lists the binary operation symbols and the commands to typeset them. The commands `\lhd`, `\rhd`, `\unlhd`, and `\unrhd` are provided by the `amssymb` package. **FIXME:** `mathdesign` doesn't properly typeset `\unlhd`.

$\amalg$ <code>\amalg</code>	$\diamond$ <code>\diamond</code>	$\setminus$ <code>\setminus</code>
$*$ <code>\ast</code>	$\div$ <code>\div</code>	$\sqcap$ <code>\sqcap</code>
$\bigcirc$ <code>\bigcirc</code>	$\triangleleft$ <code>\triangleleft</code>	$\sqcup$ <code>\sqcup</code>
$\bigtriangledown$ <code>\bigtriangledown</code>	$\mp$ <code>\mp</code>	$\star$ <code>\star</code>
$\bigtriangleup$ <code>\bigtriangleup</code>	$\odot$ <code>\odot</code>	$\triangleleft$ <code>\triangleleft</code>
$\bullet$ <code>\bullet</code>	$\ominus$ <code>\ominus</code>	$\triangleright$ <code>\triangleright</code>
$\cap$ <code>\cap</code>	$\oplus$ <code>\oplus</code>	$\triangleleft$ <code>\triangleleft</code>
$\cdot$ <code>\cdot</code>	$\oslash$ <code>\oslash</code>	$\uplus$ <code>\uplus</code>
$\circ$ <code>\circ</code>	$\otimes$ <code>\otimes</code>	$\unrhd$ <code>\unrhd</code>
$\cup$ <code>\cup</code>	$\pm$ <code>\pm</code>	$\vee$ <code>\vee</code>
$\dagger$ <code>\dagger</code>	$\rhd$ <code>\rhd</code>	$\wedge$ <code>\wedge</code>
$\ddagger$ <code>\ddagger</code>	$\times$ <code>\times</code>	$\wr$ <code>\wr</code>

`\mathsf{sans serif + abc^2}`

This typesets its argument in ‘math sans serif’: `sans serif + abc2`. ☐

`\mathtt{teletype + abc^2}`

This typesets its argument in ‘math monospaced font’. These fonts are also known as teletype fonts: `teletype + abc2`. ☐

`\mathcal{CALLIGRAPHIC}`

This typesets its argument in ‘math calligraphic’. The calligraphic letters only come in uppercase: *C A L L I G R A P H I C*. ☐

## 8.23 Symbol Tables

This section presents various tables with commands math mode symbols. Section 8.23.1 starts by presenting commands for operator symbols. This is followed by Section 8.23.2, which presents commands for relation symbols. Section 8.23.3 continues by presenting commands for arrows. Section 8.23.4 presents the remaining commands. The presentation is mainly based on [Lamport, 1994] and [Pakin, 2005].

### 8.23.1 Operation Symbols

$\TeX$  provides several symbols for binary operations. Table 8.10 lists them all.

### 8.23.2 Relation Symbols

The symbols for relations you get with  $\TeX$  is quite impressive. Table 8.11 lists  $\TeX$ ’s built-in symbols for binary relations. Additional commands which are provided by `amsmath` are listed in Table 8.12.

### 8.23.3 Arrows

$\TeX$  defines several commands for drawing arrows. All these commands produce fixed-size arrows. Extensible arrows are provided by additional packages. Table 8.13 lists all  $\TeX$ ’s built-in commands for fixed-size

**Table 8.11**

This table lists relation symbols and the commands to typeset them. The commands `\Join`, `\sqsubset`, and `\sqsupset` are provided by the `amssymb` package.

$<$	$<$	$=$	$=$	$\leq$	<code>\leq</code>
$>$	$>$	$\ll$	<code>\ll</code>	$\smile$	<code>\smile</code>
$\approx$	<code>\approx</code>	$ $	<code>\mid</code>	$\sqsubseteq$	<code>\sqsubseteq</code>
$\asymp$	<code>\asymp</code>	$\models$	<code>\models</code>	$\sqsubset$	<code>\sqsubset</code>
$\bowtie$	<code>\bowtie</code>	$\neq$	<code>\neq</code>	$\sqsupseteq$	<code>\sqsupseteq</code>
$\cong$	<code>\cong</code>	$\ni$	<code>\ni</code>	$\sqsupset$	<code>\sqsupset</code>
$\dashv$	<code>\dashv</code>	$\notin$	<code>\notin</code>	$\subseteq$	<code>\subseteq</code>
$\doteq$	<code>\doteq</code>	$\parallel$	<code>\parallel</code>	$\subset$	<code>\subset</code>
$\equiv$	<code>\equiv</code>	$\perp$	<code>\perp</code>	$\succ$	<code>\succ</code>
$\frown$	<code>\frown</code>	$\preceq$	<code>\preceq</code>	$\succ$	<code>\succ</code>
$\geq$	<code>\geq</code>	$\prec$	<code>\prec</code>	$\supseteq$	<code>\supseteq</code>
$\gg$	<code>\gg</code>	$\propto$	<code>\propto</code>	$\supset$	<code>\supset</code>
$\in$	<code>\in</code>	$\simeq$	<code>\simeq</code>	$\vdash$	<code>\vdash</code>
$\Join$	<code>\Join</code>				
$\sim$	<code>\sim</code>				

**Table 8.12**

Additional `amsmath`-provided relation symbols.

$\approx$	<code>\approxeq</code>	$\eqcirc$	<code>\eqcirc</code>	$\succapprox$	<code>\succapprox</code>
$\backepsilon$	<code>\backepsilon</code>	$\fallingdotseq$	<code>\fallingdotseq</code>	$\succcurlyeq$	<code>\succcurlyeq</code>
$\backsimeq$	<code>\backsimeq</code>	$\multimap$	<code>\multimap</code>	$\succsim$	<code>\succsim</code>
$\backsim$	<code>\backsim</code>	$\pitchfork$	<code>\pitchfork</code>	$\therefore$	<code>\therefore</code>
$\because$	<code>\because</code>	$\precapprox$	<code>\precapprox</code>	$\thickapprox$	<code>\thickapprox</code>
$\between$	<code>\between</code>	$\preccurlyeq$	<code>\preccurlyeq</code>	$\thicksim$	<code>\thicksim</code>
$\bumpeq$	<code>\bumpeq</code>	$\prec\sim$	<code>\prec\sim</code>	$\varpropto$	<code>\varpropto</code>
$\bumpeq$	<code>\bumpeq</code>	$\risingdotseq$	<code>\risingdotseq</code>	$\Vdash$	<code>\Vdash</code>
$\circeq$	<code>\circeq</code>	$\shortmid$	<code>\shortmid</code>	$\Vdash$	<code>\Vdash</code>
$\curlyeqprec$	<code>\curlyeqprec</code>	$\shortparallel$	<code>\shortparallel</code>	$\Vdash$	<code>\Vdash</code>
$\curlyeqsucc$	<code>\curlyeqsucc</code>	$\smallfrown$	<code>\smallfrown</code>	$\doteqdot$	<code>\doteqdot</code>
$\smallsmile$	<code>\smallsmile</code>				

arrows. Some commands for extensible arrows are listed in Tables 8.14–8.16. These commands, some of which accept an optional argument, require additional packages.

### 8.23.4 Miscellaneous Symbols

Table 8.17 lists  $\text{\LaTeX}$ 's “miscellaneous” symbols. It is worthwhile pointing out that the command `\imath` and `\jmath` produce a dotless  $i$  and a dotless  $j$ . These symbols should be used in combination with hats and similar decorations. The following example, should show why.

Some people write  $\hat{i} + \hat{j}$  but I prefer  $\hat{i} + \hat{j}$ .  *$\text{\LaTeX}$  Input*

The following is the output.

Some people write  $\hat{i} + \hat{j}$  but I prefer  $\hat{i} + \hat{j}$ .  *$\text{\LaTeX}$  Output*

**Table 8.13**  
Fixed-size arrow symbols.

$\downarrow$	<code>\downarrow</code>	$\Downarrow$	<code>\Downarrow</code>
$\uparrow$	<code>\uparrow</code>	$\Uparrow$	<code>\Uparrow</code>
$\updownarrow$	<code>\updownarrow</code>	$\Updownarrow$	<code>\Updownarrow</code>
$\leftarrow$	<code>\leftarrow</code>	$\Leftarrow$	<code>\Leftarrow</code>
$\rightarrow$	<code>\rightarrow</code>	$\Rightarrow$	<code>\Rightarrow</code>
$\longleftarrow$	<code>\longleftarrow</code>	$\Longleftarrow$	<code>\Longleftarrow</code>
$\longrightarrow$	<code>\longrightarrow</code>	$\Longrightarrow$	<code>\Longrightarrow</code>
$\leftrightarrow$	<code>\leftrightarrow</code>	$\Leftrightarrow$	<code>\Leftrightarrow</code>
$\longleftrightarrow$	<code>\longleftrightarrow</code>	$\Longleftrightarrow$	<code>\Longleftrightarrow</code>
$\mapsto$	<code>\mapsto</code>	$\hookrightarrow$	<code>\hookrightarrow</code>
$\longmapsto$	<code>\longmapsto</code>	$\hookleftarrow$	<code>\hookleftarrow</code>
$\leftharpoonup$	<code>\leftharpoonup</code>	$\nearrow$	<code>\nearrow</code>
$\leftharpoondown$	<code>\leftharpoondown</code>	$\searrow$	<code>\searrow</code>
$\rightharpoonup$	<code>\rightharpoonup</code>	$\swarrow$	<code>\swarrow</code>
$\rightharpoondown$	<code>\rightharpoondown</code>	$\nwarrow$	<code>\nwarrow</code>
$\rightleftharpoons$	<code>\rightleftharpoons</code>		

**Table 8.14**  
Extensible amsmath-provided  
arrow symbols.

$\overset{e}{\leftarrow}$	<code>\xleftarrow{e}</code>	$\overset{e}{\leftarrow[o]}$	<code>\xleftarrow[o]{e}</code>
$\overset{e}{\rightarrow}$	<code>\xrightarrow{e}</code>	$\overset{e}{\rightarrow[o]}$	<code>\xrightarrow[o]{e}</code>
$\overset{e}{\longleftrightarrow}$	<code>\underleftrightarrow{e}</code>	$\overset{e}{\longleftrightarrow}$	<code>\underrightarrow{e}</code>
$\overset{e}{\longleftrightarrow}$	<code>\overleftrightarrow{e}</code>	$\overset{e}{\longleftrightarrow}$	<code>\underleftrightarrow{e}</code>

**Table 8.15**  
This table lists non-standard  
mathtools-provided exten-  
sible arrow symbols and the  
commands to typeset them.  
All these commands also take  
an optional argument. The  
versions with options are  
listed in Table 8.16.

$\overset{e}{\leftrightharpoons}$	<code>\xleftrightharpoons{e}</code>	$\overset{e}{\rightharpoonleft}$	<code>\xrightleftharpoons{e}</code>
$\overset{e}{\leftharpoonup}$	<code>\xleftharpoonup{e}</code>	$\overset{e}{\rightharpoonright}$	<code>\xrightleftharpoonup{e}</code>
$\overset{e}{\leftharpoondown}$	<code>\xleftharpoondown{e}</code>	$\overset{e}{\rightharpoonleft}$	<code>\xrightleftharpoonleft{e}</code>
$\overset{e}{\rightarrow}$	<code>\xrightarrow{e}</code>	$\overset{e}{\rightarrow}$	<code>\xrightarrow{e}</code>
$\overset{e}{\hookrightarrow}$	<code>\xhookrightarrow{e}</code>	$\overset{e}{\hookrightarrow}$	<code>\xhookrightarrow{e}</code>
$\overset{e}{\hookleftarrow}$	<code>\xhookleftarrow{e}</code>	$\overset{e}{\hookleftarrow}$	<code>\xhookleftarrow{e}</code>
$\overset{e}{\mapsto}$	<code>\xmapsto{e}</code>		

It is interesting to point out that it is easier to distinguish the symbol ‘ $\ell$ ’ ( $\$ \backslash e11 \$$ ) from the digit ‘1’ than it is to distinguish the letter ‘ $l$ ’ ( $\$ l \$$ ) from the digit ‘1’. This makes ‘ $\backslash e11$ ’ an ideal alternative for the letter ‘ $l$ ’.

**Table 8.16**

This table lists non-standard `mathtools`-provided extensible arrow symbols and the commands to typeset them. Table 8.15 lists how these commands work without the optional argument.

$\xleftrightarrow{o}$	<code>\xleftrightharpoons[o]{e}</code>	$\xleftrightarrow{o}$	<code>\xrightleftharpoons[o]{e}</code>
$\xleftarrow{o}$	<code>\xleftharpoonupdown[o]{e}</code>	$\xrightarrow{o}$	<code>\xrightharpoonupdown[o]{e}</code>
$\xleftarrow{o}$	<code>\xleftharpoonup[o]{e}</code>	$\xrightarrow{o}$	<code>\xrightharpoonup[o]{e}</code>
$\longleftrightarrow$	<code>\xleftrightharpoonup[o]{e}</code>	$\longleftrightarrow$	<code>\xLeftrightharpoonup[o]{e}</code>
$\hookleftarrow$	<code>\xhookleftarrow[o]{e}</code>	$\hookrightarrow$	<code>\xhookrightarrow[o]{e}</code>
$\leftarrow$	<code>\xLeftarrow[o]{e}</code>	$\Rightarrow$	<code>\xRightarrow[o]{e}</code>
$\xrightarrow{o}$	<code>\xmapsto[o]{e}</code>		

**Table 8.17**

This table lists miscellaneous math mode symbols and the commands to typeset them. The commands `\Box`, `\Diamond`, and `\mho` are provided by the `amssymb` package.

$\aleph$	<code>\aleph</code>	$\flat$	<code>\flat</code>	$\neg$	<code>\neg</code>
$\angle$	<code>\angle</code>	$\forall$	<code>\forall</code>	$\Re$	<code>\Re</code>
$\backslash$	<code>\backslash</code>	$\hbar$	<code>\hbar</code>	$\surd$	<code>\surd</code>
$\bot$	<code>\bot</code>	$\heartsuit$	<code>\heartsuit</code>	$\top$	<code>\top</code>
$\Box$	<code>\Box</code>	$\Im$	<code>\Im</code>	$\triangle$	<code>\triangle</code>
$\clubsuit$	<code>\clubsuit</code>	$\imath$	<code>\imath</code>	$\partial$	<code>\partial</code>
$\Diamond$	<code>\Diamond</code>	$\infty$	<code>\infty</code>	$\prime$	<code>\prime</code>
$\diamondsuit$	<code>\diamondsuit</code>	$\jmath$	<code>\jmath</code>	$\sharp$	<code>\sharp</code>
$\ell$	<code>\ell</code>	$\mho$	<code>\mho</code>	$\spadesuit$	<code>\spadesuit</code>
$\emptyset$	<code>\emptyset</code>	$\nabla$	<code>\nabla</code>	$\wp$	<code>\wp</code>
$\exists$	<code>\exists</code>	$\natural$	<code>\natural</code>	$\ $	<code>\ </code>

# CHAPTER 9

## Algorithms and Listings

ALGORITHMS ARE UBIQUITOUS in computer science papers. Knowing how to present your algorithms increases the chances of getting your ideas across. The remainder of this chapter explains how to typeset pseudo-code with the `algorithm2e` package and how to present verbatim program listings with the `listings` package.

### 9.1 Typesetting Algorithms with `algorithm2e`

This section provides an introduction to `algorithm2e`, which appears to be one of the more popular packages for typesetting algorithms. The remainder of this section explains the more important aspects of the package. The content is mainly based on the package documentation [Firio, 2004]. Notice that if you don't like the `algorithm2e` package then you can always fall back to the `tabbing` environment, which is explained in Section 2.14.6.

#### 9.1.1 Importing `algorithm2e`

Importing `algorithm2e` properly may save time when writing your algorithms. An important option is `algo2e`. This option renames the environment `algorithm` to `algorithm2e` so as to avoid name clashes with other packages. There are several options which affect the appearance of the algorithms. The following three control the typesetting of blocks.

**`noline`** This option results in blocks which are typeset without lines marking the duration of the block with vertical lines. The picture to the left of Figure 9.1 demonstrates the effect of this option for a simple conditional statement.

**`lined`** This option results in vertical lines indicating the duration of the blocks. The keyword which indicates the end of the block is still typeset. The picture in the centre of Figure 9.1 demonstrates the effect of this option for a simple conditional statement.

**`vlined`** This option also result in vertical lines indicating the duration of a block. However, this time the end of each block is indicated

**Figure 9.1**

The effect of the options `noline`, `lined`, and `vlined` of `algorithm2e`. The picture to the left is the result of using

the option `noline`, that in the centre is the result of using the option `lined`, and that to the right is the result of using the option `vlined`. The option `vlined` is the most efficient in terms of saving vertical space.

---

<code>if &lt;cond&gt; then</code> <code>&lt;stuff&gt;</code> <code>end</code>	<code>if &lt;cond&gt; then</code> <code>  &lt;stuff&gt;</code> <code>end</code>	<code>if &lt;cond&gt; then</code> <code> _ &lt;stuff&gt;</code>
---	---	--

---

by a little “bend” in the line. With this option the keyword indicating the end of the block is not typeset. The picture to the right of Figure 9.1 demonstrates the effect of this option for a simple conditional statement. Compared to the other options, this option is more economical in terms of saving vertical space. When writing a paper this may make the difference between making the pagecount and overrunning it.

The `algorithm2e` has many more options, but the ones mentioned before appear to be the more useful ones. For further information the reader may wish to read the package’s documentation. All examples in the remainder of this section are typeset with the option `vlined`.

### 9.1.2 Basic Environments

The `algorithm2e` package defines a number of basic environments. Each of them is typeset in a floating environment like, which is an environment like `figure` or `table`. The `\caption` option is available in the body of the environment and works as expected. The `\caption` command is explained in Section 6.5. The command `\listofalgorithms` may be used to output a list of the algorithms with a caption. This is usually done in the document preamble. The package option `dotocloa` adds an entry for the list of algorithms in the table of contents. The following are the environments:

**algorithm** Typesets its body as an algorithm.

**algorithm\*** Typesets its body as an algorithm in a two-column document. The resulting output occupies two columns.

**procedure** Typesets its body as an procedure. Compared to `algorithm` there are a couple of differences:

- The caption starts by listing ‘Procedure `<name>`’.
- The caption must start with ‘`<name>`( `<arguments>` )’.

**procedure\*** Typesets its body as an procedure in a two-column document. This environment works just as `procedure` but the resulting output occupies two columns.

**function** Typesets its body as a function. This environment works just as `procedure`.



**Figure 9.2**  
Using `algorithm2e`.

---

<pre>\begin{algorithm2e}[H] \KwIn{   Integers <math>a \geq 0</math> and <math>b \geq 0</math>}  \KwOut{\textsc{Gcd} of <math>a</math> and <math>b</math>} \While{<math>b \neq 0</math>}{   <math>r \leftarrow a \bmod b</math>;   <math>a \leftarrow b</math>;   <math>b \leftarrow r</math>; } \caption{Euclidean Algorithm} \end{algorithm2e}</pre>	<p><b>Input:</b> Integers  <math>a \geq 0</math> and  <math>b \geq 0</math></p> <p><b>Output:</b> GCD of <math>a</math>  and <math>b</math></p> <p><b>while</b> <math>b \neq 0</math> <b>do</b>    <math>r \leftarrow a \bmod b</math>;    <math>a \leftarrow b</math>;    <math>b \leftarrow r</math>;</p> <p><b>Algorithm 1:</b> Euclidean Algorithm</p>
---	--

---

**function\*** Typesets its body as a function in a two-column document. This environment works just as function but the resulting output occupies two columns.

Each environment can be positioned using the optional argument of the environment. As usual the optional argument is any combination of ‘p’, ‘t’, ‘b’, or ‘h’, and each has the usual meaning. This positioning mechanism is explained in Section 6.5. The option ‘H’ is also allowed and means “definitely here”. If you don’t know how to use these optional positioning arguments then it is recommended that you use ‘tbp’:

```
\begin{algorithm2e}[tbp]
...
\end{algorithm2e}
```

L<sup>A</sup>T<sub>E</sub>X Usage

It is always a good to get some idea of the functionality of a package by looking at an example. Figure 9.2 demonstrates some of the functionality of `algorithm2e`. Notice that the semicolons are typeset with the command `\;`.

### 9.1.3 Describing Input and Output

The `algorithm2e` package defines several commands for describing the input and output of the algorithms. It also provides a mechanism to add keywords and define a style for classes of keywords. This section briefly mentions the main commands for describing the input and output of the algorithms.

`\KwIn{⟨input⟩}`

This typesets the value of the ‘In’ label followed by `⟨input⟩`. Figure 9.2 demonstrates how this works. It is possible to redefine the value of the label for ‘In’. This is also possible for all other labels mentioned in this list. □

`\KwOut{⟨input⟩}`

This typesets the value of the ‘Out’ label followed by `⟨output⟩`. □

`\KwData{⟨input⟩}`

This typesets the value of the ‘Data’ label followed by `⟨input⟩`. ☐

`\KwResult{⟨output⟩}`

This typesets the value of the ‘Result’ label followed by `⟨output⟩`. ☐

`\KwRet{⟨value⟩}`

This typesets the value of the ‘Ret’ label followed by `⟨value⟩`. The command is used to describe return values. ☐

### 9.1.4 Conditional Statements

The `algorithm2e` package defines a large array of commands for typesetting conditional statements. This includes commands for typesetting one-line statements. The remainder of this section explains some of the commands for typesetting simple multi-line conditional statements. Information about the remaining commands may be found in the the package documentation. The following are the commands.

`\If(⟨comment⟩){⟨condition⟩}{⟨clause⟩}`

This typesets a single conditional statement with condition `⟨condition⟩` and final then clause `⟨clause⟩`. The argument which is enclosed in parentheses is for describing a comment. This argument is optional and may be omitted (including the arguments). The following is an example of the resulting output. The comment has been omitted.

<code>\If{⟨condition⟩}</code>	<b>if</b> <code>⟨condition⟩</code> <b>then</b>	<input type="checkbox"/>
<code>⟨clause⟩</code>	<code> </code> <code>⟨clause⟩</code>	

`\uIf(⟨comment⟩){⟨condition⟩}{⟨clause⟩}`

This works as `\If` only this time it is assumed that `⟨clause⟩` is not the final clause. The following is the resulting output.

<code>\uIf{⟨condition⟩}{</code>	<b>if</b> <code>⟨condition⟩</code> <b>then</b>	<input type="checkbox"/>
<code>⟨clause⟩}</code>	<code> </code> <code>⟨clause⟩</code>	

`\ElseIf(⟨comment⟩){⟨condition⟩}{⟨clause⟩}`

This typesets a conditional else clause with condition `⟨condition⟩` and final if else clause `⟨clause⟩`.

<code>\ElseIf{⟨condition⟩}</code>	<b>else if</b> <code>⟨condition⟩</code> <b>then</b>	<input type="checkbox"/>
<code>{⟨clause⟩}</code>	<code> </code> <code>⟨clause⟩</code>	

`\uElseIf(⟨comment⟩){⟨condition⟩}{⟨clause⟩}`

This typesets a conditional else clause with condition `⟨condition⟩` and non-final else clause `⟨clause⟩`.

<code>\eElseIf{⟨condition⟩}</code>	<b>else if</b> <code>⟨condition⟩</code> <b>then</b>	<input type="checkbox"/>
<code>{⟨clause⟩}</code>	<code> </code> <code>⟨clause⟩</code>	

**Figure 9.3**

Typesetting conditional statements with `algorithm2e`.

---

<pre>\begin{algorithm2e}[tbp] \If{\$a &lt; 0\$}{   \tcp{\$a &lt; 0\$} } \uElseIf{\$a = 0\$}{   \tcp{\$a = 0\$} } \lElse\eIf{\$a = 1\$}{   \tcp{\$a = 1\$} } {   \tcp{\$a &gt; 1\$} } \end{algorithm2e}</pre>	<pre>if <math>a &lt; 0</math> then   // <math>a &lt; 0</math> else if <math>a = 0</math> then   // <math>a = 0</math> else if <math>a = 1</math> then   // <math>a = 1</math> else   // <math>a &gt; 1</math></pre>
--	---

---

`\eIf(<comment>){<condition>}{<then clause>}{<comment>}{<else clause>}}`

This typesets the if else clause with condition `<condition>` with then clause `<then clause>` and final else clause `<else clause>`. As suggested by the notation, both `<comment>` arguments are optional.

<pre>\eIf{&lt;condition&gt;}{   &lt;then clause&gt; }{&lt;then clause&gt;}</pre>	<pre>if &lt;condition&gt; then   &lt;then clause&gt; else   &lt;else clause&gt;</pre>	□
--	---	---

`\lElse`

This typesets the word `else`. This is mainly useful in combination with `\eIf`. □

Figure 9.3 provides an example which demonstrates how to typeset a complex-ish if statement. The command `\tcp` typesets its argument as a C++ comment.

### 9.1.5 The Switch Statement

This section briefly explains `algorithm2e`'s commands for typesetting switch statements. The following are the commands.

`\Switch(<comment>){<value>}{<cases>}}`

This typesets the first line and the braces for the body of the switch statement. The following is the resulting output.

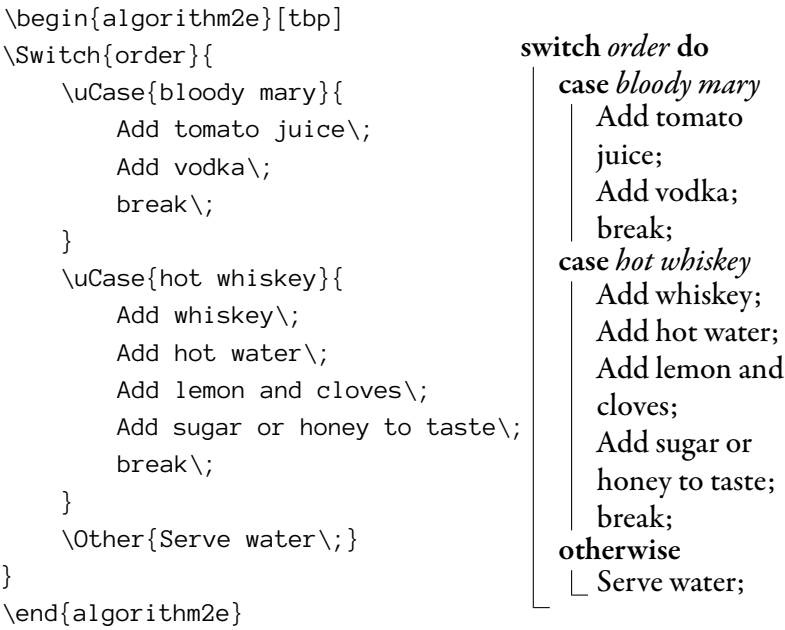
<pre>\Switch{&lt;value&gt;}{   &lt;cases&gt;}</pre>	<pre>switch &lt;value&gt; do   &lt;cases&gt;</pre>	□
---	--	---

`\Case(<comment>){<condition>}{<statements>}}`

This typesets the final case of the switch statement. The following is the resulting output.

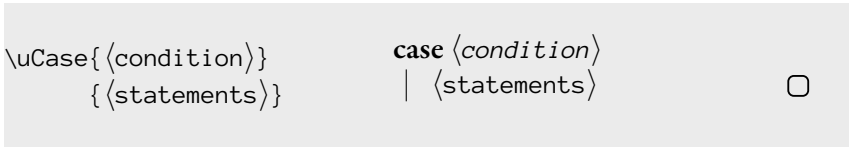
<pre>\Case{&lt;condition&gt;}{   &lt;statements&gt;}</pre>	<pre>case &lt;condition&gt;   &lt;statements&gt;</pre>	□
--	--	---

**Figure 9.4**  
Using algorithm2e’s switch  
statements.



`\uCase(<comment>){<condition>}{<statements>}`

This also typesets a case of the switch statement, but here it is assumed the case is not the last case of the switch statement. The following is the resulting output.



`\Other(<comment>){<statements>}`

This typesets the default case of the switch statement. The following is the resulting output.

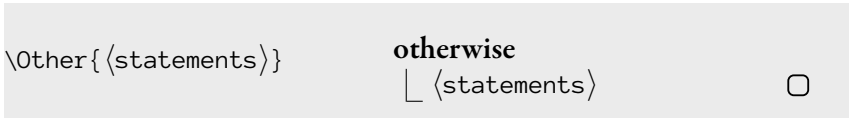


Figure 9.4 provides a complete example of how to typeset a switch statement.

9.1.6 Iterative Statements

The algorithm2e package has constructs for several iterative statements, including while, for, foreach-based, and repeat-until statements. This section provides a brief explanation of each of these commands.

The following are the commands.

`\For(<comment>){<condition>}{<body>}`

This typesets a basic for statement with a “condition” `<condition>` and body `<body>`. The following is an example of the result.

<code>\For{⟨condition⟩}</code> <code>{⟨body⟩}</code>	<b>for</b> <i>⟨condition⟩</i> <b>do</b>   <i>⟨body⟩</i>	□
---	--	---

`\ForEach(⟨comment⟩){⟨condition⟩}{⟨body⟩}`

This typesets a foreach statement with a “condition” *⟨condition⟩* and body *⟨body⟩*. The following is an example of the result.

<code>\ForEach{⟨condition⟩}</code> <code>{⟨body⟩}</code>	<b>foreach</b> <i>⟨condition⟩</i> <b>do</b>   <i>⟨body⟩</i>	□
---	--	---

`\ForAll(⟨comment⟩){⟨condition⟩}{⟨body⟩}`

This typesets a forall statement with a “condition” *⟨condition⟩* and body *⟨body⟩*. The following is an example of the result.

<code>\ForAll{⟨condition⟩}</code> <code>{⟨body⟩}</code>	<b>forall</b> <i>⟨condition⟩</i> <b>do</b>   <i>⟨body⟩</i>	□
--	---	---

`\While(⟨comment⟩){⟨condition⟩}{⟨body⟩}`

This typesets a while statement with condition *⟨condition⟩* and body *⟨body⟩*. The following is an example of the result.

<code>\While{⟨condition⟩}</code> <code>{⟨body⟩}</code>	<b>while</b> <i>⟨condition⟩</i> <b>do</b>   <i>⟨body⟩</i>	□
---	--	---

`\Repeat(⟨comment⟩){⟨condition⟩}{⟨body⟩}(⟨comment⟩)`

This typesets a repeat-until statement with and body *⟨body⟩*. The following is an example of possible output.

<code>\Repeat{⟨condition⟩}</code> <code>{⟨body⟩}</code>	<b>repeat</b>   <i>⟨body⟩</i> <b>until</b> <i>⟨condition⟩</i> ;
--	---

Note that the first comment is put on the repeat line, whereas the second comment is put on the until line. □

### 9.1.7 Comments

This Section, which concludes the discussion of the `algorithm2e` package, explains how to typeset comments. Comments are defined in a C and C++ style. For a given language there are different styles of comments. The command for typesetting C comments is `\tcc`, that for typesetting C++ comments is called `tcp`. The following explains the `tcp` command. The `\tcc` command works analogously.

`\tcp{⟨comment⟩}`

Typesets the comment *⟨comment⟩*. The comment may consist of several lines, which should be separated with the newline command (`\`). The following is an example.

```
\tcp{⟨line one⟩}\ \           // ⟨line one⟩
      {⟨line two⟩}           // ⟨line two⟩           □
```

`\tcp*{⟨comment⟩}`

This typesets the side comment `⟨comment⟩` right justified. The command `\tcp*[r]{⟨comment⟩}` works analogously.

```
⟨statement⟩
  \tcp*{⟨comment⟩}           ⟨statement⟩; // ⟨comment⟩ □
```

`\tcp*[l]{⟨comment⟩}`

This typesets the side comment `⟨comment⟩` left justified.

```
⟨statement⟩
  \tcp*[l]{⟨comment⟩}       ⟨statement⟩; // ⟨comment⟩ □
```

`\tcp*[h]{⟨comment⟩}`

This typesets the comment `⟨comment⟩` left justified in place (here).

```
\If(\tcp*[h]{⟨comment⟩})
  {⟨condition⟩}           if ⟨condition⟩ then // ⟨comment⟩
  {⟨statement⟩}           ⏟ ⟨statement⟩           □
```

`\tcp*[f]{⟨comment⟩}`

This typesets the comment `⟨comment⟩` right justified in place (here).

```
\If(\tcp*[f]{⟨comment⟩})
  {⟨condition⟩}           if ⟨condition⟩ then // ⟨comment⟩
  {⟨statement⟩}           ⏟ ⟨statement⟩           □
```

## 9.2 Typesetting Listings with the `listings` Package

The `listings` package is one of the nicer packages for creating formatted output. The remainder of this section is a brief example-driven introduction to the package. More information may be found in the package documentation [Heinz and Moses, 2007].

The `listings` package supports typesetting of listings. The package provides support for several languages, including ANSI C, and ANSI C++, Eiffel, HTML, Java, PHP, Python, L<sup>A</sup>T<sub>E</sub>X, and XML. The package supports user-defined styles for keywords and identifiers. Two methods are provided for specifying a listing.

**Environment** An environment called `lstlisting` for specifying a listing in the body of the environment.

**Command** A command called `\lstinputlisting` for creating a listing from a source file. The required argument of this command is the name of the source code file. The optional argument is for setting the options.

Both the environment and the command take an optional argument, in the form of a `⟨key⟩=⟨value⟩` list, for overriding the default settings. The

**Figure 9.5**  
Creating a partial listing with  
the listings package.

---

```
\begin{lstlisting}[language=Java%
                    ,gobble=3%
                    ,numbers=left%
                    ,firstline=2%
                    ,lastline=4%
                    ,firstnumber=2%
                    ,caption=Hello World.%
                    ,label=example]

public class Greetings {
    \lllpublic static void main( String[] args ) {
    \lll    System.out.println( "Hello world!" );
    \lll}
}
\end{lstlisting}
```

---

**Figure 9.6**  
Listing resulting from Fig-  
ure 9.5

---

```
2 \public static void main( String[] args ) {
3 \    System.out.println( "Hello world!" );
4 \}
```

---

Listing 1. Hello world.

---

package also provides a command for setting new defaults. The resulting algorithm may be typeset at the current position or as a floating algorithm with a number and caption. The package also provides a command called `\listoflistings` for typesetting a list of numbered listings.

Figure 9.5 shows how you use the `lstlistings` environment. The resulting output is presented in Figure 9.6. Note that not all of the body of the environment is typeset and that the resulting numbers are generated automatically. The following explains the relevant options.

language

This specifies the programming language. Possible values are ‘C’, ‘[ANSI]C’, ‘C++’, ‘[ANSI]C++’, ‘HTML’, ‘Eiffel’, ‘HTML’, ‘Java’, ‘PHP’, ‘Python’, ‘LaTeX’, and ‘XML’. ☐

gobble

This determines the number of characters which are removed from the start of each line in the input. The default value is 0. ☐

numbers

This is used to control the placement of numbers. Possible values are ‘none’ (default) for no numbers, ‘left’ for numbers to the left, and ‘right’ for numbers to the right. ☐

firstline

The value of this option determines the number of the first input line that is typeset. It may be useful to skip a number of lines at the start of the source. The default value is 1. ☐

lastline

This option determines the number of the last input line that is typeset. The default value is the number of lines in the input. ☐

**Figure 9.7**  
Setting new defaults with the  
`\lstset` command.

---

```
\lstset{language=Java%
        ,keywordstyle=\bfseries\ttfamily%
        ,stringstyle=\ttfamily%
        ,identifierstyle=\ttfamily\itshape%
        ,showspaces=false%
        ,showstringspaces=true%
        ,numbers=left%
        ,float%
        ,floatplacement=tbp%
        ,captionpos=b}
```

---

firstnumber

This is the first line number in the output. ☐

caption

This determines the caption of the typeset listing. ☐

label

This determines the label of the typeset listing. ☐

As already announced the `listings` package also provides a command for specifying new default option values. The name of this command is `\lstset` and its required argument is a `<key>=<value>` list argument specifies the new default values for the options.

Figure 9.7 provides an example that overrides some of the default settings. Some of these options have been explained before. The remaining options work as follows.

keywordstyle

The value of this option should be a series of declarations that determines how keywords are typeset. The declarations `'\bfseries\ttfamily'` in Figure 9.7 result in bold face keywords which are typeset in a monospaced font. ☐

stringstyle

The value of this option should be a series of declarations that determines how characters are typeset in strings. The declaration `'\ttfamily'` in Figure 9.7 result in string characters which are typeset in a monospaced font. ☐

identifierstyle

The value of this option should be a series of declarations that determines how identifiers are typeset. The declarations `'\ttfamily\itshape'` in Figure 9.7 result in identifiers which are typeset in a monospaced italic font. ☐

showspaces

If the value of this option is `'true'` then spaces are typeset as visual spaces. The default value is `'false'`. ☐

showstringspaces

If the value of this option is `'true'` then spaces in strings are typeset as visual spaces. The default value is `'false'`. ☐



float	If this option is provided then the listing is typeset as a float.	<input type="checkbox"/>
floatplacement	The value of this option determines the float placement. It can be any sequence of characters in 'tbph'.	<input type="checkbox"/>
captionpos	This determines the position of the caption. Possible values are 't' (top) and 'b' (bottom).	<input type="checkbox"/>



# **Part V**

## **Automation**



# CHAPTER 10

## Commands and Environments

THIS CHAPTER STUDIES user-defined commands and environments. Section 10.1 starts by studying advantages and disadvantages of commands. This is followed by Section 10.2, which explains how to define user-defined commands. Section 10.3 recalls the working of  $\text{\TeX}$ 's four processors and Section 10.4 explains how they process  $\text{\LaTeX}$  commands. Section 10.5 explains how to define commands in plain  $\text{\TeX}$ . Section 10.6 presents a common technique for tweaking existing commands. Section 10.7 presents a technique which overcomes the problem that  $\text{\LaTeX}$  allows no more than nine arguments. Section 10.8 is an introduction to environments and Section 10.9 concludes by explaining how to define your own environments.

### 10.1 Why use Commands

$\text{\LaTeX}$  is a programmable typesetting engine. Commands are the key to controlling your document. The advantages of using commands in  $\text{\LaTeX}$  are similar to the advantages of using functions and procedures in high-level programming languages. However,  $\text{\LaTeX}$  commands also have disadvantages. We shall first study advantages and then disadvantages. The following are some advantages.

**Software engineering** Tedious tasks can be automated. This has the following advantages.

**Reusability** Commands which are defined once can be reused several times.

**Simplicity** Carrying out a complex task using a simple command with a well-understood interface is much easier and leads to fewer errors.

**Refinement** You can stepwise refine the implementation of certain tasks. This allows you to postpone certain decisions. For example, if you haven't been able to decide how to typeset certain symbols which serve a certain purpose, then you may start typesetting them using a command which typesets them in a simple manner. This lets you start writing the

document in terms of high-level notions. By refining the command at a later stage, you can fine-tune the typesetting of all the relevant symbols.

**Maintainability** This advantage is related to the previous item. Unforeseen changes in requirements can be implemented easily by making a few local changes.

**Consistency** Typesetting entities using carefully chosen commands guarantees a consistent appearance of your document. For example, if you typeset your pseudo-code identifiers using a pseudo-code identifier typesetting command in a ‘pseudo-code identifier’ style, then your identifiers will have a consistent feel.

**Computing** Tasks and results may be computed depending on document options. This has the following advantages.

**Style control** Things may be typeset in a style which depends on class or package options. For example, the `article` class typesets the main text in 10 pt by default but providing a 12pt option gives you a 12 pt size.

**Content control** Commands may result in different output if there are different global options. For example, consider the `beamer` class, which allows you to prepare a computer presentation and lecture notes in the same input. It provides options which allow you to hide certain parts of your lecture notes in the presentation and vice versa. This is very a potent feature as it allows sharing and guarantees consistency between the notes and the presentation.

**Typeset results** This issue is related to the previous item.  $\text{\LaTeX}$  can do basic arithmetic, can branch and iterate, and can typeset the *results* of computations. For example, the `lipsum` package provides a command `\lipsum[⟨number1⟩–⟨number2⟩]` which typesets the ‘Lorem ipsum’ Paragraphs  $\langle \text{number}_1 \rangle$  to  $\langle \text{number}_2 \rangle$  and including. You can easily extend this command to make it repeat the paragraphs  $N$  times. As another example, again consider the `beamer` class. It allows you to generate several pages for your presentation from a single frame environment. Within the frame you may have a itemised list whose items are uncovered, one at a time, in your presentation. The uncovering results in several partial and one final page for the single frame. As a final example, the `calctab` package provides the basic functionality of a spreadsheet with computation rules for output columns in tables.

The following are some disadvantages of  $\text{\LaTeX}$  commands, most of which are inherited from  $\text{\TeX}$ .

**Number of arguments** T<sub>E</sub>X sadly does not allow more than nine arguments per macro. It may be argued that commands which require more than nine arguments are not well-designed, but this does not make the restriction less arbitrary.

**Numbers as arguments** This disadvantage is probably the source of the previous disadvantage. When implementing T<sub>E</sub>X, Knuth decided to refer to formal arguments of macros as numbers. The first is called #1, the second is called #2, and so on. Needless to say that this makes it extremely easy for T<sub>E</sub>X to parse and recognise arguments, but this prevents programmers from giving meaningful names to the arguments, makes it difficult to understand the implementation of the commands, and makes it easy to make mistakes.

**Flat namespace** T<sub>E</sub>X allows local definitions at the group level but its namespace is flat at the top level. As a consequence all the commands which are defined at the top level are global. This is arguably the greatest problem. With thousands of packages and classes this requires that package and class implementors have to be careful to avoid name clashes.

## 10.2 User-defined Commands

This section studies command definitions. Section 10.2.1 explains how to define and redefine commands that take no arguments. Section 10.2.2 explains how to define and redefine commands that do take arguments, and Section 10.2.3 explains the difference between *fragile* and *robust* commands. Section 10.2.4 explains how to define robust commands and make existing commands robust.

### 10.2.1 Defining Commands Without Arguments

ƎT<sub>E</sub>X has several ways to define new commands. The following are for defining and redefining commands which take no arguments.

`\newcommand⟨cmd⟩{⟨subst⟩}`

This defines a new command, ⟨cmd⟩, with substitution text ⟨subst⟩. In T<sub>E</sub>X parlance ⟨cmd⟩ is called a *command sequence*. A ƎT<sub>E</sub>X command sequence starts with a backslash and is followed by a non-empty sequence of symbols — usually letters. The new command does not take any arguments. The substitution text ⟨subst⟩ is substituted for each occurrence of ⟨cmd⟩ which is expanded by the Expansion Processor. This does not include all occurrences. For example ⟨cmd⟩ is not expanded if it occurs in the substitution text of other ƎT<sub>E</sub>X definitions at definition time. A more detailed description of the expansion of ƎT<sub>E</sub>X commands is provided in Section 10.4. □

`\renewcommand⟨cmd⟩{⟨subst⟩}`

This redefines the command ⟨cmd⟩, which should be an existing command. The resulting command has substitution text ⟨subst⟩ and does

not take any arguments. □

The following is an example of a  $\text{\LaTeX}$  program which defines a user-defined command  $\backslash\text{CTAN}$  and uses it in the body of the document environment.

$\text{\LaTeX}$  Usage

```

\documentclass{article}
\newcommand{\CTAN}{Comprehensive \TeX{} Archive Network}
\begin{document}
  I always download my packages from the \CTAN.
  The \CTAN{} is the place to be.
\end{document}

```

The substitution text of the command is ‘Comprehensive  $\text{\TeX}$ {} Archive Network’. Given this definition  $\text{\LaTeX}$  substitutes the substitution text ‘Comprehensive  $\text{\TeX}$ {} Archive Network’ for  $\backslash\text{CTAN}$  each time  $\backslash\text{CTAN}$  is used. The following is the resulting output.

$\text{\LaTeX}$  Output

I always download my packages from the Comprehensive  $\text{\TeX}$  Archive Network. The Comprehensive  $\text{\TeX}$  Archive Network is the place to be.

## 10.2.2 Defining Commands With Arguments

Defining commands with arguments is done in a similar way. The following are the relevant commands for defining commands without optional arguments.

$\backslash\text{newcommand}\langle\text{cmd}\rangle[\langle\text{digit}\rangle]\{\langle\text{subst}\rangle\}$

As before, this defines a new command,  $\langle\text{cmd}\rangle$ , with substitution text  $\langle\text{subst}\rangle$ . This time the command takes  $\langle\text{digit}\rangle$  arguments. The number of arguments should be in the range 1–9. The  $i$ -th formal argument is referred to as  $\#i$  in the substitution text  $\langle\text{subst}\rangle$ . When substituting  $\langle\text{subst}\rangle$  for  $\langle\text{cmd}\rangle$   $\text{\TeX}$ ’s Expansion Processor also substitutes the  $i$ -th actual argument for  $\#i$  in  $\langle\text{subst}\rangle$ , for  $1 \leq i \leq \langle\text{digit}\rangle$ . It is not allowed to use  $\#i$  in  $\langle\text{subst}\rangle$  if  $i < 1$  or  $\langle\text{digit}\rangle < i$ . □

$\backslash\text{renewcommand}\langle\text{cmd}\rangle[\langle\text{digit}\rangle]\{\langle\text{subst}\rangle\}$

This redefines  $\langle\text{cmd}\rangle$  as a command with  $\langle\text{digit}\rangle$  arguments and substitution text  $\langle\text{subst}\rangle$ . □

The standard way to define a command with an optional argument is as follows. By default the optional argument can only be used in the first position.

$\backslash\text{newcommand}\langle\text{cmd}\rangle[\langle\text{digit}\rangle][\langle\text{default}\rangle]\{\langle\text{subst}\rangle\}$

This defines a new command sequence,  $\langle\text{cmd}\rangle$ , with substitution text  $\langle\text{subst}\rangle$ . As before the command takes  $\langle\text{digit}\rangle$  arguments. However, this time the first argument ( $\#1$ ) is optional. If present it should be enclosed in square brackets. If the optional argument is omitted then it is assigned the value  $\langle\text{default}\rangle$ . □

The command  $\backslash\text{renewcommand}$  may also be used to define commands with optional arguments. This is done as follows:  $\backslash\text{renewcommand}\langle\text{cmd}\rangle$



**Figure 10.1**  
User-defined commands.

---

```

\usepackage{multind}
\makeindex{command}
\makeindex{package}

\newcommand{\MonoIdx}[2][command]{
  \texttt{#2}%
  \index{#1}{\texttt{#2}}%
}

\begin{document}
...The command
  \MonoIdx{\textbackslash MakeRobustCommand}
  is provided by the package
  \MonoIdx[package]{makerobust}. ...
\printindex{command}{Index of Commands}
\printindex{package}{Index of Packages}
\end{document}

```

---

[ $\langle \text{digit} \rangle$ ][ $\langle \text{default} \rangle$ ]{ $\langle \text{subst} \rangle$ }.

The  $\text{\LaTeX}$  program which is depicted in Figure 10.1 uses multiple index files and defines a user-defined command `\MonoIdx` which typesets its second argument in monospaced font and writes information about it to these index files. The optional argument is used to determine the name of the index file.

### 10.2.3 Fragile and Robust Commands

Having dealt with advantages and disadvantages of  $\text{\LaTeX}$  commands and knowing how to define them, we're ready to study *fragile* and *robust* commands. The reason for studying them is that they are a common cause of errors, which are caused by command side-effects. To make things worse these errors may occur in subsequent  $\text{\LaTeX}$  sessions and at seemingly unrelated locations. These errors are difficult to deal with — especially for novice users. Some of these issues are related to the notions of *moving arguments* and *fragile* and *robust* commands. The remainder of this section explains how to deal with fragile commands in moving arguments and avoid these common errors.

A *moving argument* of a command is saved by the command to be reread later on. Examples of moving arguments are arguments which appear in the Table of Contents, in the Table of Figures, in indexes, and so on. For example, the `\caption` command which defines captions of tables and figures writes these captions to the list of tables (`.lot`) and list of figures (`.lof`) files respectively. The list of tables and list of figures files are reread when  $\text{\LaTeX}$  typesets the list of figures and the list of tables.

Moving arguments are expanded before they are saved. Sometimes the expansion leads to invalid  $\text{\TeX}$  being written to a file. When this invalid  $\text{\TeX}$  is reread in a subsequent session this may cause errors.

A command is called *robust* if it does not expand to valid T<sub>E</sub>X. Otherwise it is called *fragile*.

The command `\protect` protects commands against expansion. If `\protect\command` is saved then this saves `\command`. This allows you to protect fragile commands in moving arguments. In effect this postpones the expansion of `\command` until it is reread.

### 10.2.4 Defining Robust Commands

The following commands are related to defining robust commands and making existing commands robust.

`\DeclareRobustCommand⟨cmd⟩{⟨subst⟩}`

This defines `⟨cmd⟩` as a robust command without arguments and substitution text `⟨subst⟩`. □

`\DeclareRobustCommand⟨cmd⟩[⟨digit⟩]{⟨subst⟩}`

This defines `⟨cmd⟩` as a robust command with substitution text `⟨subst⟩` and `⟨digit⟩` arguments. □

`\DeclareRobustCommand⟨cmd⟩[⟨digit⟩][⟨default⟩]{⟨subst⟩}`

This defines `⟨cmd⟩` as a robust command with substitution text `⟨subst⟩` and `⟨digit⟩` arguments, one of which is optional with default value `⟨default⟩`. □

`\MakeRobustCommand⟨cmd⟩`

This turns the existing command `⟨cmd⟩` into a robust command. `\MakeRobustCommand` is not a standard command but is provided by the package `makerobust`. □

## 10.3 The T<sub>E</sub>X Processors

Before studying how L<sup>A</sup>T<sub>E</sub>X expands (evaluates) commands, this section briefly revisits the four processors which T<sub>E</sub>X is built upon. It is recalled from Section 1.2.1 that these processors are run in a pipeline. The following describes them. The following description is based on [Eijkhout, 2007, Chapter 1].

**Input Processor** The Input Processor turns T<sub>E</sub>X's input stream into a token stream, which is sent to the Expansion Processor.

**Expansion Processor** The Expansion Processor turns its input token stream into a token stream of non-expandable tokens. Among others, the Expansion Processor is responsible for *macro expansion* (command expansion) and *decision making*. The resulting stream is sent to the Execution Processor.

**Execution Processor** The Execution Processor executes its input sequentially from start to finish. Tasks which are carried out by the Execution Processor are state-affecting assignments to T<sub>E</sub>X registers (variables) and the construction of horizontal, vertical, and math lists. The resulting output lists are sent to the Visual Processor.

**Visual Processor** The Visual Processor does paragraph breaking, alignment, page breaking, mathematical typesetting, and `.dvi` generation. The final output is the `.dvi` file.

## 10.4 Commands and Arguments

This section explains how  $\text{\TeX}$  applies commands to arguments. Throughout this section it is assumed that the input stream has been tokenised by  $\text{\TeX}$ 's Input Processor. At this stage there are two kinds of tokens:

**Character tokens** A character token represents a single character in the input.

**Control sequence tokens** Control sequence tokens correspond to commands. They represent a sequence of characters in the input starting with a backslash and continuing with a sequence of other tokens.

$\text{\TeX}$ 's Expansion and Execution Processors can distinguish between the character and control sequence tokens, which makes it easy to recognise tokens which correspond to commands.

It remains to explain how  $\text{\TeX}$  parses arguments. This is slightly more difficult. There are two kinds of arguments, which we shall refer to as *primitive* and *compound* arguments.

**Primitive** Simple arguments consist of a single character or control sequence token. The tokens of the opening and closing brace are not allowed.

**Compound** A compound argument corresponds to a brace-delimited group in the input. The token at the start of the group is that of an opening brace (`{`) and that at the end of the group is that of a closing brace (`}`). Within the sequence brace pairs should be balanced. Most of the time you will use compound arguments. The value of a compound argument is the sequence of tokens “in” the group, that is the sequence of tokens *without* the tokens of its (first) opening brace and that of its (last) closing brace [Knuth, 1990, pages 204–205]. For example, given a command `\single` that takes one single argument, the actual parameter of `\single{lab{c}}` is given by `lab{c}`.

The remainder of this section provides examples of command expansion. We shall start with a simple example which involves primitive arguments only, and continue with a more complex example which involves both primitive and compound arguments.

The following should explain what is going on with primitive arguments. Let's assume we have two user-defined commands called `\swop` and `\SWOP` which are defined as follows.

**Figure 10.2**

A program with user-defined combinators.

---

```

\documentclass{article}

\newcommand\K[2]{#1}
\newcommand\S[3]{#1#3{#2#3}}
\newcommand\I{\S\K\K}
\newcommand\X{\S{\K{\S\I}}{\S{\K\K}\I}}

\begin{document}
  \X abc
\end{document}

```

---

```

\newcommand\swop[2]{#2#1}
\newcommand\SWOP[2]{#2#1}

```

*ℒ<sub>TEX</sub> Usage*

Both commands do the same but, for sake of the example, they've been given different names. They take two arguments and 'output' (rewrite them to) the second argument followed by the first. Having defined these commands, '`\swop2\SWOP31`' now give us '321'. To see what has happened, notice that the first argument of the command `\swop` is the character token which corresponds to the character '2' and notice that the second argument is the command sequence token which corresponds to the '`\SWOP`' in the input. Expanding '`\swop3\SWOP`' reverses the order of the arguments giving us the token sequence '`\SWOP231`'. Expanding this token sequence gives us '321', which is completely expanded, cannot be expanded any further, and completes the rewriting process.

The following is a more complex example. Let's assume we have the  $\text{\LaTeX}$  program which is listed in Figure 10.2. The program defines four commands `\K`, `\S`, `\I`, and `\X`. The first three commands correspond to the combinators  $K$ ,  $S$ , and  $I$  from Moses Schönfinkel and Haskell Curry's combinatory logic. They may be describes as follows:  $K\langle A \rangle \langle B \rangle \mapsto \langle A \rangle$ ,  $S\langle A \rangle \langle B \rangle \langle C \rangle \mapsto \langle A \rangle \langle C \rangle (\langle B \rangle \langle C \rangle)$ , and  $I \mapsto SKK$ . If you study the  $\text{\TeX}$  definition of the command `\X` you may notice that it does not have any formal arguments. It may therefore come as a surprise that it correspond to a combinator,  $X$ , which swops its arguments, i.e.  $X\langle A \rangle \langle B \rangle \mapsto \langle B \rangle \langle A \rangle$ . Still this makes perfect sense and the remainder of this section explains why.

Knowing that `\X` corresponds to a combinator which swops its arguments we should be able to predict the output of our program — it should be 'bac.' Let's see if we can explain this properly. Table 10.1 illustrates the expansion process. The second column of the table lists the output of the Expansion Processor, the third column lists the current input stream of the Expansion Processor, and the first column lists the number of the reductions. The subscripts of the tokens in the input stream correspond to the nesting level of the groups.

The first reduction is that of `\X` to its substitution text. It does not involve any argument. Reduction 2 is a reduction of the form  $\S\langle A \rangle \langle B \rangle \langle C \rangle \mapsto \langle A \rangle \langle C \rangle \{ \langle B \rangle \langle C \rangle \}$ , where  $\langle A \rangle$  and  $\langle B \rangle$  correspond to the

Table 10.1

$\text{\TeX}$ 's Expansion Processor. The output and the input of the Expansion Processor are listed in the second and third column. The numbers of the reductions are listed in the first column. Each token in the input has a subscript which corresponds to the nesting-level of groups.

#	Out	In
1		$\backslash X_1 a_1 b_1 c_1$
2		$\backslash S_1 \{ \backslash K_2 \{ \backslash S_3 \backslash I_3 \}_2 \}_1 \{ \backslash S_2 \{ \backslash K_3 \backslash K_3 \}_2 \backslash I_2 \}_1 a_1 b_1 c_1$
3		$\backslash K_1 \{ \backslash S_2 \backslash I_2 \}_1 a_1 \{ \backslash S_2 \{ \backslash K_3 \backslash K_3 \}_2 \backslash I_2 a_2 \}_1 b_1 c_1$
4		$\backslash S_1 \backslash I_1 \{ \backslash S_2 \{ \backslash K_3 \backslash K_3 \}_2 \backslash I_2 a_2 \}_1 b_1 c_1$
5		$\backslash I_1 b_1 \{ \backslash S_2 \{ \backslash K_3 \backslash K_3 \}_2 \backslash I_2 a_2 b_2 \}_1 c_1$
6		$\backslash S_1 \backslash K_1 \backslash K_1 b_1 \{ \backslash S_2 \{ \backslash K_3 \backslash K_3 \}_2 \backslash I_2 a_2 b_2 \}_1 c_1$
7		$\backslash K_1 b_1 \{ \backslash K_2 b_2 \}_1 \{ \backslash S_2 \{ \backslash K_3 \backslash K_3 \}_2 \backslash I_2 a_2 b_2 \}_1 c_1$
8		$b_1 \{ \backslash S_2 \{ \backslash K_3 \backslash K_3 \}_2 \backslash I_2 a_2 b_2 \}_1 c_1$
9	b	$\{ \backslash S_2 \{ \backslash K_3 \backslash K_3 \}_2 \backslash I_2 a_2 b_2 \}_1 c_1$
10	b	$\backslash S_2 \{ \backslash K_3 \backslash K_3 \}_2 \backslash I_2 a_2 b_2 \}_1 c_1$
11	b	$\backslash K_2 \backslash K_2 a_2 \{ \backslash I_3 a_3 \}_2 b_2 \}_1 c_1$
12	b	$\backslash K_2 \{ \backslash I_3 a_3 \}_2 b_2 \}_1 c_1$
13	b	$\backslash I_2 a_2 \}_1 c_1$
14	b	$\backslash S_2 \backslash K_2 \backslash K_2 a_2 \}_1 c_1$
15	b	$\backslash K_2 a_2 \{ \backslash K_3 a_3 \}_2 \}_1 c_1$
16	b	$a_2 \}_1 c_1$
17	ba	$\}_1 c_1$
18	ba	$c_1$
		bac

top-level groups in the input and  $\langle c \rangle$  corresponds to the character token which represents the lower case letter ‘a.’ Removing the opening and closing brace tokens of the groups and applying the reduction gives us the input of Reduction 3. The third reduction is of the form  $\backslash K \langle A \rangle \langle B \rangle \mapsto \langle A \rangle$  where both  $\langle A \rangle$  and  $\langle B \rangle$  are groups. Removing the second group, removing the opening and closing brace tokens of the first group, and applying the reduction gives us the input of Reduction 4. All remaining reductions are similar except for Reduction 9 and 17, which correspond to entering a group and leaving the group. The final result is listed in the last row. It should give confidence that the output is ‘bac’ as expected.

## 10.5 Defining Commands with $\text{\TeX}$

In this section we shall study how to define  $\text{\LaTeX}$  commands using plain  $\text{\TeX}$ .  $\text{\TeX}$  allows a richer variety of commands than  $\text{\LaTeX}$ . The main differences are that  $\text{\TeX}$  commands come in local and global flavours. In addition they may be defined with *delimiters* in their argument list. Usually, you should not need  $\text{\TeX}$  command definitions but sometimes they are needed. The best thing to do is define commands using  $\text{\LaTeX}$  commands and only define commands with  $\text{\TeX}$  as a final resort.

The following are  $\text{\TeX}$ 's commands for defining commands without delimiters.

```
\def <cmd>#1#2...#n{ <subst> }
```

This defines a command,  $\langle \text{cmd} \rangle$ , with  $n$  arguments, and with substitution

text  $\langle\text{subst}\rangle$ . The command is local to the group in which it is defined. The numbers in the formal parameter list must contain the numbers 1– $n$ , in increasing order. This restriction holds for all T<sub>E</sub>X command definitions.  $\square$

$\backslash\text{edef}\langle\text{cmd}\rangle\#1\#2\dots\#n\{\langle\text{subst}\rangle\}$

This defines a command,  $\langle\text{cmd}\rangle$ , with  $n$  arguments, and substitution text which is the *expansion* of  $\langle\text{subst}\rangle$ . It should be noted that  $\langle\text{subst}\rangle$  is expanded at the time at which  $\langle\text{cmd}\rangle$  is defined. The command is local to the group in which it is defined.  $\square$

The following should explain how the commands  $\backslash\text{def}$  and  $\backslash\text{edef}$  work. Given the definitions in the following listing ‘ $\backslash\text{hello}\{\}\backslash\text{ehello}$ ’ gives us ‘HI hi’.

```
 $\backslash\text{def}\backslash\text{hi}\{\text{hi}\}$ 
 $\backslash\text{def}\backslash\text{hello}\{\backslash\text{hi}\}$ 
 $\backslash\text{edef}\backslash\text{ehello}\{\backslash\text{hi}\}$ 
 $\backslash\text{def}\backslash\text{hi}\{\text{HI}\}$ 
```

*L<sup>A</sup>T<sub>E</sub>X Usage*

Commands which are defined using  $\backslash\text{def}$  or  $\backslash\text{edef}$  are not allowed to have paragraph breaks. To allow paragraph breaks in arguments you add the prefix  $\backslash\text{long}$  to  $\backslash\text{def}$  or  $\backslash\text{edef}$ .

As stated in the explanation of T<sub>E</sub>X macro definitions, commands may be defined locally in a group. What is more, they may also be defined locally within other macro definitions. Formal parameters in macro definitions which are nested inside other definitions receive an extra ‘#’ character to distinguish them from the formal parameters of the nesting macro definition(s). Using this mechanism and the definition in the following listing ‘ $\backslash\text{sillyo!}$ ’ gives us ‘!0!1!’.

```
 $\backslash\text{def}\backslash\text{silly}\#1\#2\{\%$ 
     $\backslash\text{def}\backslash\text{sillier}\##1\{!\##1!\#2!\}\backslash\text{sillier}\{\#1\}\%$ 
 $\}$ 
```

*L<sup>A</sup>T<sub>E</sub>X Input*

The following commands are useful when defining low-level commands with T<sub>E</sub>X.

$\backslash\text{csname}\backslash\langle\text{tokens}\rangle\backslash\text{endcsname}$

This results in the command sequence of the expansion of  $\langle\text{tokens}\rangle$ . In effect this expands  $\langle\text{tokens}\rangle$  and puts a backslash character to the front of the result. For example,  $\backslash\text{csname}\backslash\text{command}\backslash\text{endcsname}$  gives  $\backslash\text{command}$ . To see why expansion matters, let’s assume we have the definition  $\backslash\text{def}\backslash\text{ho}\{\text{Ho}\}$ . With this definition  $\backslash\text{csname}\backslash\text{Ho}\backslash\text{ho}\backslash\text{ho}\backslash\text{endcsname}$  gives us  $\backslash\text{HoHoHo}$ .  $\square$

$\backslash\text{noexpand}\langle\text{token}\rangle$

This results in  $\langle\text{token}\rangle$  without expanding it. For example, the definitions of the commands  $\backslash\text{def}\backslash\text{hello}\{\backslash\text{hi}\}$  and  $\backslash\text{edef}\backslash\text{hello}\{\backslash\text{noexpand}\backslash\text{hi}\}$  are equivalent regardless of the definition of  $\backslash\text{hi}$ .  $\square$

$\backslash\text{expandafter}\langle\text{token}\rangle\langle\text{tokens}\rangle$

This expands the first token in  $\langle\text{tokens}\rangle$  (using arguments if required) and inserts  $\langle\text{token}\rangle$  before the result.  $\square$

**Figure 10.3**  
Defining commands with default arguments.

---

```
% allow @ in macro names
\makeatletter%
\def\cmd#1{%
  \ifnextchar[%
    % use the given option
    {\cmd@relay{#1}}%
    % use the default option
    {\cmd@relay{#1}[dflt]}%
  }
\def\cmd@relay#1[#2]{...}
% disallow @ in macro names
\makeatother
```

---

```
\makeatletter
\def\cmd#1{%
  \def\cmd@relay##1[##2]{...}
  \ifnextchar[%
    {\cmd@relay{#1}}%
    {\cmd@relay{#1}[dflt]}%
  }
\makeatother
```

---

The command `\expandafter` is frequently used in combination with `\csname` to construct definitions with parameterised names. The following example demonstrates this mechanism. The `\expandafter` allows `\csname` and `\endcsname` construct the command sequence name before applying the `\def` command.

```
\documentclass{article}
\def\property#1{%
  \expandafter\def%
  \csname#1\endcsname##1{%
    ##1\ is #1}%
}
\property{brilliant}
\property{excellent}
\begin{document}
  \excellent{\TeX} and
  \brilliant{\LaTeX}.
\end{document}
```

TeX is excellent and L<sup>A</sup>TeX is brilliant.

TeX also allows commands with delimiters in argument lists. For example, it lets you implement a command `\command` which uses the character ‘|’ to delimit its two arguments. This allows you to apply the command to one and two by writing `\command|one|two|`. Using TeX you define a command like this as follows.

```
\def\command|#1|#2|{...}
```

L<sup>A</sup>T<sub>E</sub>X Usage

More complex delimiters are also allowed. For example, combinations of letters, spaces, and command sequences are valid delimiters, even if the command sequences do not correspond to existing commands. It is also not required that all arguments be delimited or that all delimiters be equal.

Figure 10.3 provides two different implementations of a contrived command which has one default argument. In L<sup>A</sup>TeX terms the example defines a user-defined L<sup>A</sup>TeX command which takes two parameters. The *second* argument is optional with default ‘dflt.’

Let's first study the solution to the left. There are two new aspects to this solution. The first is the use of the commands `\makeatletter` and `\makeatother`. The command `\makeatletter` allows '@' symbols in command names. The command `\makeatother` disallows them. This is a common idiom as it lets you — with high probability — define command sequences which are unique. The second new aspect is the use of `\ifnextchar` `<character>` `<first>` `<second>` which looks ahead to see if the next character is equal to `<character>` without consuming it. It results in `<first>` if the next character is `<character>` and results in `<second>` otherwise. In the solution to the left the user-defined command `\cmd` looks ahead to see the token following the first argument, and passes control to the command `\cmd@relay` with the proper option.

The solution to the right is similar but it defines the relay command locally. It is recalled that formal parameters of nested macro definitions receive extra '#' characters. Therefore, the formal parameters of `\cmd@relay` are now `##1` and `##2`. Using this mechanism should allow you to refer to both the formal parameters of `\cmd` and the formal parameters of `\cmd@relay` inside the substitution text of `\cmd@relay`.

Candidate delimiters inside matching brace pairs are ignored. For example, let's assume we have the following definition.

<code>\def\agoin{ old chap}</code>	<i>LaTeX Usage</i>
<code>\def\hows#1\agoin{How are you #1?}</code>	

Then `'\hows{Joe\agoin}\agoin'` gives 'How are you Joe old chap?'

## 10.6 Tweaking Existing Commands with `\let`

This section studies how to tweak existing commands, i.e. redefine an existing command in such a way that it carries out an additional task. To do this we are going to use TeX's `\let` command by assigning the meaning of the original command to a scratch command sequence. Next we redefine the existing command and refer to the scratch command sequence when we want to carry out the task which was associated with the original command. In the example in Figure 10.4 we redefine the `\section` command and force it to take one more argument, which is the label of the section. The resulting command first uses the original `\section` command to define the section and next uses the `\label` command to define the label.

## 10.7 More than Nine Arguments

As mentioned in Section 10.1, TeX does not allow you to have more than nine arguments. This section describes two techniques which help you to overcome this problem. Both techniques exploit the fact that TeX allows local definitions of commands.

To illustrate the solutions we shall implement a command `\command` which takes ten arguments and outputs their values. The first technique



**Figure 10.4**

A section unit environment.

---

```

\makeatletter
% Save meaning of old \section command.
\let\old@section=\section
\def\section#1#2{%
  % Define section using old \section command.
  \old@section{#2}
  % Define label for the section.
  \label{#1}
}
\makeatother

```

---

**Figure 10.5**

Using more than nine arguments.

---

```

\makeatletter
\def\cmd#1#2#3#4#5#6#7#8#9{%
  \def\cmd@arg@A{#1}%
  \def\cmd@arg@B{#2}%
  :
  \def\cmd@arg@I{#9}%
  \relay%
}
\def\relay#1{%
  \def\cmd@arg@J{#1}%
  Arguments: \cmd@arg@A, \cmd@arg@B, ..., and \cmd@arg@J.%
}
\makeatother

```

---

is to implement `\command` as a wrapper command which does two things.

- It formally defines nine local commands. The  $i$ -th local command results in the value of the  $i$ -th argument of `\command`.
- It passes control to a ‘relay’ function which can see the remaining argument.

Figure 10.5 demonstrates the technique. The second technique is simpler and implements `\relay` as a local macro. The following demonstrates the technique.

```

\def\cmd#1#2#3#4#5#6#7#8#9{%
  \def\relay##1{Arguments: ##1, ##2, ..., and #1.}%
  \relay%
}
\makeatother

```

L<sup>A</sup>T<sub>E</sub>X Usage

## 10.8 Introduction to Environments

This section is about environments. The following are a few reasons in favour of environments.

**Figure 10.6**

User-defined environment.

---

```

\newcommand{\endOfSectionCommand}{...}
\newenvironment{SectionalUnit}[2][section]
    {\csname#1\endcsname{#2}}
    {\endOfSectionCommand}

\begin{document}
  \begin{SectionalUnit}[chapter]{Introduction}
    \begin{SectionalUnit}{Conventions}
      ...
    \end{SectionalUnit}
  \begin{SectionalUnit}{Notation}
    ...
  \end{SectionalUnit}
\end{SectionalUnit}
:
\end{document}

```

---

**Declarativeness** Arguably, using an environment is more declarative than using a command.

**Less ambiguity** If commands with arguments are used as part of other commands with arguments then this may make it difficult to see which closing brace belongs to which command. If environments are used inside other environments then it is easier to see which `\begin{<env>}` belongs to which `\end{<env>}`, thereby resolving the ‘brace ambiguity’.

**Allows Paragraphs** You can have paragraphs inside environments.

**More Efficient** Environments can be implemented without the need of extra stack space. This makes their implementation more efficient than macros.

## 10.9 Environment Definitions

This section studies how to define user-defined environments. The key to defining environments is the command `\newenvironment`, which is used as follows.

```
\newenvironment{<name>}{<begin subst>}{<end subst>}
```

This defines a new global environment which is called `<name>`. When you write `\begin{<name>}{<body>}\end{<name>}` the text `<begin subst>` is substituted for `\begin{<name>}` and the text `<end subst>` is substituted for `\end{<name>}`. Effectively, this gives you `<begin subst>{<body>}<end subst>`. □

```
\newenvironment{<name>}[<digit>]{<begin subst>}{<end subst>}
```

This defines a new global environment `<name>` with `<digit>` arguments. In addition to the mechanism for environments without arguments there

is now also argument substitution. However, argument substitution only works within `\begin subst`. This works just as for commands, so the  $i$ -th actual argument of the environment is substituted for the  $i$ -th formal argument,  $\#i$ , in `\begin subst`. It is not allowed to refer to formal arguments in `\end subst`. □

`\newenvironment{<name>}[<digit>][<default>]{<begin subst>}{<end subst>}`

This defines a new global environment which is called `<name>` and takes `<digit>` arguments, the first of which is optional. □

The command `\renewenvironment` is for redefining environments. It works as ‘expected’.

Figure 10.6 presents an example of a user-defined environment which takes two arguments, one of which is optional. It is left as an exercise to the reader to determine how the resulting environment works.



# CHAPTER 11

## Option Parsing

THIS SHORT CHAPTER DISCUSSES two packages for implementing ‘ $\langle\text{key}\rangle=\langle\text{value}\rangle$ ’ macro interfaces. They overcome several problems with  $\text{\LaTeX}$ ’s argument mechanism. Using this technique you can implement a command called `\figure` that takes optional arguments which describe a rotation angle and a scale for the resulting figure. The resulting command may be used as `\figure[angle=90,scale=2]{mypicture.pdf}`. We shall first study the more rudimentary `keyval` package. Next we shall continue studying `keycommand` package, which is more recent and much easier to use. The main reasons for studying the `keyval` package is that it is used a lot, and that studying it provides some insight in what is required to implement the required functionality. Before studying the packages, we shall study the motivation for using ‘ $\langle\text{key}\rangle=\langle\text{value}\rangle$ ’ interfaces.

### 11.1 Why Use a $\langle\text{Key}\rangle=\langle\text{Value}\rangle$ Interface?

We’ve already seen that  $\text{\LaTeX}$ ’s argument handling mechanism is not ideal. The following are some arguments in favour of  $\langle\text{key}\rangle=\langle\text{value}\rangle$  interfaces.

**Number of arguments** There is no limit to the number of arguments.

**Robustness** The mechanism is more robust. The arguments can be supplied in any order. For example, ‘`\compare[apples=4,oranges=5]`’ and ‘`\compare[oranges=5,apples=4]`’ should do the same. Default values can be defined for missing arguments.

**Interface** By relating the value to the key, the purpose of the argument is clear. This makes the interface clearer and easier to use.

**Names** The mechanism reduces references to the meaningless formal parameter names ‘`#1`’, ‘`#2`’, .... Instead it allows the programmer to get the value of a specific key.

### 11.2 The `keyval` Package

At the time of writing the `keyval` package [D. Carlisle, 1999b] is one of the more commonly used packages for implementing  $\langle\text{key}\rangle=\langle\text{value}\rangle$

interfaces.

To study the `keyval` package we shall implement a contrived command `\compares[apples=<apples>,oranges=<oranges>]{<name>}`. The task of the command is to typeset the text ‘<name> compares <apples> apples with <oranges> oranges’. The first argument of the command should be truly optional. In addition, the command should be flexible/robust: the order of the `<key>=<value>` pairs shouldn’t matter and it shouldn’t be required to list them all. The default value for `<apples>` is 2 and the default value for `<oranges>` is 3. So ‘`\compares[apples=9]{Mary}`’ should result in the text ‘Mary compares 9 apples with 3 oranges’ and ‘`\compares[oranges=2,apples=2]{Peter}`’ should result in ‘Peter compares 2 apples with 2 oranges’. Throughout we shall assume that the `@` symbol is allowed in command sequence names.

We start by importing the `keyval` package and by defining the default values. Next we use the `\define@key{<family>}{<key>}{<action>}` command to inform `keyval` about the existence of the key `apples` and the key `oranges`. The command `\define@key` is provided by the `keyval` package. Its `<family>` argument tells `keyval` about the *family* of keys. Here the *family* corresponds to the keys for our specific application. By introducing different families, you can use the same key in different families but with different rules for dealing with the key. The `<key>` argument of `\define@key` specifies the name of the key, and the `<action>` specifies what to do with the value for the given `<key>`. Inside the `<action>` argument, `#1` represents the actual value for the given `<key>` in an actual `<key>=<value>` list. In both cases we let the `<action>` parameter override the default value for `<key>`.

```
\usepackage{keyval}
\def\compares@apples{2}
\def\compares@oranges{3}
\define@key{compares}{apples}%
    {\def\compares@apples{#1}}
\define@key{compares}{oranges}%
    {\def\compares@oranges{#1}}
```

*LaTeX Usage*

Having informed `keyval` about the keys and what to do with them, the rest is straightforward. The following listing defines our command `\compares`. All it does is insert an empty `<key>=<value>` list if there is no `<key>=<value>` list and forward control to a command called `\@compares` which does the actual work. We used a similar technique as on Page 203 in Section 10.5. The command `\@compares` is relatively straightforward. It starts by parsing the `<key>=<value>` pairs in the square bracket-delimited argument. This is done with the `\setkeys` command, which is provided by `keyval`. Having determined the `<value>`s for the `<key>`s, all that remains is the typesetting.

```

\def\compares{%
  \@ifnextchar[%
    {\@compares}%
    {\@compares[]}}
\def\@compares[#1]#2{%
  {\setkeys{compares}{#1}%
   #2\ compares \compares@apples~apples
   with \compares@oranges~oranges.}}

```

*LaTeX Usage*

Note the extra group within the `\@compares` command. Its main purpose is keeping the re-definitions of the keys local. An alternative solution is to use local macros to define the default values of the keys and then use `\setkeys` to assign the provided values.

### 11.3 The keycommand Package

This section studies a recent alternative to the `keyval` package: the `keycommand` package. Essentially, `keycommand` provides a high-level mechanism for defining macros and environments with  $\langle \text{key} \rangle = \langle \text{value} \rangle$  interfaces. The following are the building blocks.

```
\newkeycommand{\command}[\langle \text{key-value list} \rangle][\langle \text{number} \rangle]{\langle \text{definition} \rangle}
```

This defines a new command  $\langle \text{command} \rangle$  that takes  $\langle \text{number} \rangle$  *regular* arguments and one optional argument and substitution text  $\langle \text{definition} \rangle$ . The optional argument is a list of  $\langle \text{key} \rangle = \langle \text{value} \rangle$  pairs, the keys and default values of which are listed in  $\langle \text{key-value list} \rangle$ . For each key  $\langle \text{key} \rangle$  and default value  $\langle \text{default} \rangle$ , the argument  $\langle \text{key-value list} \rangle$  should have an entry of the form ' $\langle \text{key} \rangle = \langle \text{default} \rangle$ '. Inside the  $\langle \text{definition} \rangle$  you use `\commandkey{\langle \text{key} \rangle}` to get the actual value for the  $\langle \text{key} \rangle$ . The following implements our command `\compares`.

```

\newkeycommand{\compares}[apples=3,oranges=2][1]{%
  #1\ compares \commandkey{apples}~apples
  with \commandkey{oranges}~oranges.}

```

*LaTeX Usage*

```
\newkeyenvironment{\name}[\langle \text{key-value list} \rangle][\langle \text{number} \rangle]{\langle \text{start} \rangle}{\langle \text{end} \rangle}
```

This defines a new environment  $\langle \text{name} \rangle$  with begin and end substitution text  $\langle \text{begin} \rangle$  and  $\langle \text{end} \rangle$ . The remaining arguments are similar to the arguments of `\newkeycommand`.

The `keycommand` package also provides commands for redefining commands and environments. The reader is referred to the package documentation [Chervet, 2009] for further information.





# CHAPTER 12

## Branching

THIS CHAPTER IS DEVOTED TO decision making, and branching. The techniques in this chapter allow you to implement the equivalent of `if` and `while` clauses in  $\text{\LaTeX}$ . This gives you ultimate control over the style *and* content of your documents.

The remainder of this chapter is as follows. Section 12.1 studies counters, Boolean variables, and lengths. Section 12.2 demonstrate how to implement `if` and `while` statements with the `ifthen` package. Section 12.4 studies the use of `for` loops in  $\text{\LaTeX}$ . Section 12.5 concludes this chapter by demonstrating how to implement tail-recursion in low-level  $\text{\TeX}$ .

### 12.1 Counters, Booleans, and Lengths

This section provides an introduction to counters, Boolean variables, and length-related commands. The reason for studying them is that they play the rôle of variables in  $\text{\LaTeX}$  and  $\text{\TeX}$ .

#### 12.1.1 Counters

A  $\text{\LaTeX}$  *counter* is a global variable for counting things. The following are the commands related to  $\text{\LaTeX}$  counters.

```
\newcounter{<name>}
```

This defines a new global *counter*. There a *counter* is a  $\text{\LaTeX}$  variable that can take integer values. It is not quite clear which range is allowed for counters, except that (some) positive, (some) negative, and (all!) zero values are allowed. The initial value of the counter is zero. According to Lamport, the command `\newcounter` may not be defined in files which are `\included` [Lamport, 1994, Page 138]. You may only use the command in the document preamble [Lamport, 1994, Page 99], but I've noticed that putting it elsewhere is also allowed.  $\square$

```
\setcounter{<name>}{<value>}
```

This assigns the value `<value>` to the counter `<name>`. Here `<name>` should be the name of an existing counter and `<value>` should be an integer constant.  $\square$

```
\stepcounter{<name>}
```

This increments the counter `<name>` by one. As with `\setcounter`, `<name>`

should be the name of an existing counter. □

`\addtocounter{⟨name⟩}{⟨increment⟩}`

This adds the constant `⟨inc⟩` to the counter `⟨name⟩`. As before, `⟨name⟩` should be the name of an existing counter and `⟨value⟩` should be an integer constant. □

`\the⟨name⟩`

This gives you the value of the counter `⟨name⟩`, which should be the name of an existing counter. Here `\the⟨name⟩` is the concatenation of ‘`\the`’ and ‘`⟨name⟩`’. For example, the counter section is used in  $\text{\LaTeX}$  for counting the current section number, and the command `\thesection` gives you the number of the current section. □

`\newcounter{⟨slave⟩}[⟨master⟩]`

This defines a *slave counter* `⟨slave⟩` which depends on *master counter* `⟨master⟩`, which should be an existing counter. Here, a *slave counter* of a *master counter* is a counter which is numbered “within” the master counter. For example, the subsection counter is a slave counter of the master counter section. If `⟨master⟩` is incremented using the `\stepcounter` command, then the counter `⟨slave⟩` is automatically reset. This process also recursively resets slave counters of `⟨slave⟩`. This version of the `\newcounter` command is useful for implementing counter hierarchies. □

The following example demonstrates these counter-related commands, except for the version of `\newcounter` with the optional argument.

```

\newcounter{answer} % define answer
\setcounter{answer}{9} % assign 9 to answer.
\addtocounter{answer}{11} % add 11 to answer
\stepcounter{answer} % increment answer
\addtocounter{answer}{\theanswer} % double answer

\begin{document}
  The answer is~\theanswer.
\end{document}

```

*LaTeX Usage*

### 12.1.2 Booleans

$\text{\LaTeX}$  does not support decision making. To make decisions you need  $\text{\TeX}$  or use a package such as `ifthen`. In the remainder of this section we shall study  $\text{\TeX}$ ’s way of decision making. The `ifthen` package is studied in Section 12.2.

`\newif\if⟨bool⟩`

This is  $\text{\TeX}$ ’s way to define a branching command called `\if⟨bool⟩`. You may regard it as the definition of an artificial Boolean variable called `⟨bool⟩`. For example, you may define a Boolean “variable” ‘notes’ with the command ‘`\newif\ifnotes`’. □

`\⟨bool⟩true`

This is equivalent to assigning `true` to the Boolean “variable” `⟨bool⟩`. □

**Table 12.1**  
Length units.

Unit	Name	Equivalent
pt	point	
pc	pica	$1\text{pc} = 12\text{pt}$
in	inch	$1\text{in} = 72.27\text{pt}$
bp	big point	$72\text{bp} = 1\text{in}$
cm	centimetre	$2.54\text{cm} = 1\text{in}$
mm	millimetre	$10\text{mm} = 1\text{cm}$
dd	didot point	$1157\text{dd} = 1238\text{pt}$
cc	cicero	$1\text{cc} = 12\text{dd}$
sp	scaled point	$65536\text{sp} = 1\text{pt}$

`\<bool>false`

This is equivalent to assigning `false` to the Boolean “variable” `<bool>`. □

`\if<bool><then clause>\fi`

This is  $\text{\TeX}$ ’s equivalent of a conditional statement. As expected this results in `<then clause>` if the value of the Boolean “variable” `<bool>` is true. □

`\if<bool><then clause>\else<else clause>\fi`

This is the equivalent of an if-else statement. It results in `<then clause>` if the value of the Boolean “variable” `<bool>` is true and results in `<else clause>` otherwise. □

The following is an example that creates a section. The title of the section depends on the value of the boolean variable `notes`. If `notes` is true then the title is set to ‘Lecture Notes’. Otherwise, the section is titled ‘Presentation’. This example can be taken further to implement a context-sensitive document the style *and* content of which depends on the values of Boolean variables.

```
\newif\ifnotes
\notesttrue

\begin{document}
\section{\ifnotes Lecture Notes%
        \else Presentation%
        \fi}

...
\end{document}
```

*ETEX Usage*

### 12.1.3 Lengths

This chapter studies *length* variables, which are  $\text{\TeX}$ / $\text{\LaTeX}$  variables that can be assigned measures of distance. They are also be used for decision making. This section is mainly based on [Lamport, 1994, Section 6.4].

$\text{\TeX}$  has a wide range of length (measure) units. Table 12.1 lists them all. Each length unit represents its own length. Writing ‘`1<unit>`’ gives you the length of the unit `<unit>`. For example ‘`1mm`’ gives you

the length of one millimetre. Likewise you multiply  $\langle \text{unit} \rangle$  by any constant  $\langle \text{constant} \rangle$  by writing ‘ $\langle \text{constant} \rangle \langle \text{unit} \rangle$ ’. For example, ‘101in’ is equivalent to ‘256.54cm’.

*Length variables* hold length values. They are denoted as command sequences. Given length variable  $\langle \text{len} \rangle$ ,  $2\langle \text{len} \rangle$  gives you twice its current value.

There are two kinds of lengths: *rigid* and *rubber*. The following explains the difference between the two.

**Rigid** A rigid length which always gives you the same length.

**Rubber** A rubber length is a combination of length and elasticity. Their values may stretch or shrink depending on the situation. This is useful for stretching/shrinking inter-word space and so on. Multiplying a rubber length makes it rigid, so  $1.0\backslash\text{rubber}$  gives you a rigid length which is the equivalent of the length value of  $\backslash\text{rubber}$ .

The following are some of  $\text{\LaTeX}$ ’s length-related commands. By defining your formatting commands in terms of these commands you can make them work regardless of the current document settings.

$\backslash\text{parindent}$

This length variable stores the amount of indentation at the beginning of a normal paragraph. ☐

$\backslash\text{textwidth}$

This length variable stores the width of the text on the page. ☐

$\backslash\text{textheight}$

This length variable stores the height of the body of a page, excluding the head and foot space. ☐

$\backslash\text{parskip}$

This length variable stores the extra vertical space between paragraphs. This is a rubber length with a natural length of zero. With this setting the vertical space which is caused by  $\backslash\text{parskip}$  usually does not result in additional inter-paragraph spacing. However, it does allow the length to stretch if the  $\backslash\text{flushbottom}$  declaration is in effect. ☐

$\backslash\text{baselinekip}$

This length variable stores the vertical distance from the bottom of one line in a paragraph to the bottom of the next line in the same paragraph. ☐

The following are  $\text{\LaTeX}$ ’s length-related commands.

$\backslash\text{newlength}\{\langle \text{command} \rangle\}$

This defines the length command  $\langle \text{command} \rangle$  with an initial value of 0cm. For example, the command  $\backslash\text{newlength}\{\backslash\text{mylen}\}$  defines a new length called  $\backslash\text{mylen}$ . ☐

$\backslash\text{setlength}\{\langle \text{command} \rangle\}\{\langle \text{length} \rangle\}$

This assigns the length value  $\langle \text{length} \rangle$  to the length command  $\langle \text{command} \rangle$ . For example, the command  $\backslash\text{setlength}\{\backslash\text{parskip}\}\{1.0\text{mm}\}$  assigns the value 1mm to  $\backslash\text{parskip}$ . ☐

`\addtolength{⟨command⟩}{⟨length⟩}`

This adds the length value  $\langle\text{length}\rangle$  to the current value of the length  $\langle\text{command}\rangle$ . For example, the spell `\addtolength{\parskip}{1.0mm}` adds a millimetre to `\parskip`. □

`\settowidth{⟨command⟩}{⟨stuff⟩}`

This assigns the width of  $\langle\text{stuff}\rangle$  to  $\langle\text{command}\rangle$ . For example, the command `\settowidth{\twoms}{MM}` assigns twice the width of the text ‘MM’ to `\twoms`. □

`\settoheight{⟨command⟩}{⟨stuff⟩}`

This assigns the height of  $\langle\text{stuff}\rangle$  to  $\langle\text{command}\rangle$ . For example, the command `\settoheight{\tower}{\$2^{2^2}\$}` assigns the height of  $2^{2^2}$  to `\tower`. □

`\settodepth{⟨command⟩}{⟨stuff⟩}`

This assigns the depth of  $\langle\text{stuff}\rangle$  to  $\langle\text{command}\rangle$ . For example, the command `\settodepth{\parskip}{amazing}` sets the value of `\parskip` to the distance the letter ‘g’ extends below the line. □

The commands `\setlength` and `\addtolength` obey the normal scoping rules.

### 12.1.4 Scoping

This section briefly explains the difference between the scoping rules for assignments to counters, T<sub>E</sub>X Booleans, and lengths. Counters are *global* which is to say that the values of counter variables are *not* restored upon leaving a group. T<sub>E</sub>X Booleans and lengths satisfy *group scoping rules*, which means that upon leaving a group these variables are assigned the same values which they had upon entering the group.

## 12.2 The `ifthen` Package

This section studies the `ifthen` package, which provides the functionality of defining Boolean variables at the L<sup>A</sup>T<sub>E</sub>X level, decision making, and branching. There are two commands for defining new Boolean variables.

`\newboolean{⟨bool⟩}`

This defines a new global Boolean variable. the command will fail if  $\langle\text{bool}\rangle$  is already defined. □

`\provideboolean{⟨bool⟩}`

This also defines a new global Boolean variable. However, this command will accept  $\langle\text{bool}\rangle$  if it is already defined. □

`\setboolean{⟨bool⟩}{⟨value⟩}`

This assigns the value  $\langle\text{value}\rangle$  to  $\langle\text{bool}\rangle$ . Here  $\langle\text{value}\rangle$  should be true or false. □

Knowing how to define Boolean variables we can proceed with making decisions. The command `\ifthenelse{⟨test⟩}{⟨then clause⟩}{⟨else clause⟩}` is a two-way branching construct. As expected it carries out  $\langle\text{then clause}\rangle$  if  $\langle\text{test}\rangle$  evaluates to true and carries out  $\langle\text{else clause}\rangle$  if  $\langle\text{test}\rangle$  evaluates to false.

clause} if  $\langle \text{test} \rangle$  evaluates to false. The condition  $\langle \text{test} \rangle$  must be of the following form:

$\langle \text{boolean} \rangle$	Here $\langle \text{boolean} \rangle$ should be ‘true’ or ‘false’, ignoring case, so ‘true’, ‘trueE’, ..., ‘TRUE’, and ‘TRUE’ are equivalent, and so are ‘false’, ‘falseE’, ..., ‘FALSE’, and ‘FALSE’.	<input type="checkbox"/>
$\langle \text{number}_1 \rangle \langle \text{op} \rangle \langle \text{number}_2 \rangle$	Here $\langle \text{number}_1 \rangle$ and $\langle \text{number}_2 \rangle$ should be numbers and $\langle \text{op} \rangle$ should be ‘<’, ‘=’, or ‘>’.	<input type="checkbox"/>
$\backslash \text{lengthtest} \{ \langle \text{dimen}_1 \rangle \langle \text{op} \rangle \langle \text{dimen}_2 \rangle \}$	Here $\langle \text{dimen}_1 \rangle$ and $\langle \text{dimen}_2 \rangle$ should be dimension values and $\langle \text{op} \rangle$ should be ‘<’, ‘=’, or ‘>’.	<input type="checkbox"/>
$\backslash \text{isodd} \{ \langle \text{number} \rangle \}$	As suggested by the notation $\langle \text{number} \rangle$ should be a number.	<input type="checkbox"/>
$\backslash \text{isundefined} \{ \langle \text{command} \rangle \}$	Here $\langle \text{command} \rangle$ should be a command sequence name.	<input type="checkbox"/>
$\backslash \text{equal} \{ \langle \text{string}_1 \rangle \} \{ \langle \text{string}_2 \rangle \}$	Here $\langle \text{string}_1 \rangle$ and $\langle \text{string}_2 \rangle$ are evaluated and compared for equality. The test is equivalent to true if and only if the results of the evaluations are equal.	<input type="checkbox"/>
$\backslash \text{boolean} \{ \langle \text{bool} \rangle \}$	Here $\langle \text{bool} \rangle$ should be a Boolean variable.	<input type="checkbox"/>
$\langle \text{test}_1 \rangle \langle \text{command} \rangle \langle \text{test}_2 \rangle$	Here $\langle \text{test}_1 \rangle$ and $\langle \text{test}_2 \rangle$ should be valid $\langle \text{test} \rangle$ conditions and $\langle \text{command} \rangle$ is $\backslash \text{or}$ , $\backslash \text{and}$ , $\backslash \text{OR}$ , or $\backslash \text{AND}$ . The versions $\backslash \text{OR}$ and $\backslash \text{AND}$ are preferred to $\backslash \text{or}$ and $\backslash \text{and}$ as they are more robust.	<input type="checkbox"/>
$\langle \text{negation} \rangle \langle \text{test} \rangle$	Here $\langle \text{test} \rangle$ should be a valid $\langle \text{test} \rangle$ condition and $\langle \text{negation} \rangle$ should be ‘\not’ or ‘\NOT’. The upper case version is preferred to the lower case version.	<input type="checkbox"/>
$\backslash ( \langle \text{test} \rangle \backslash )$	Here $\langle \text{test} \rangle$ should be a valid $\langle \text{test} \rangle$ condition.	<input type="checkbox"/>

The following example demonstrates how to use the  $\backslash \text{ifthenelse}$  command. The page counter variable, which is used in the example, keeps track of L<sup>A</sup>T<sub>E</sub>X’s page numbers.

```
\usepackage{ifthen}
\begin{document}
\ifthenelse{\isodd{\value{page}}}{%
    {We're on an odd page.}%
    {The page is even.}%
\end{document}
```

L<sup>A</sup>T<sub>E</sub>X Usage

The command  $\backslash \text{whiledo} \{ \langle \text{test} \rangle \} \{ \langle \text{statement} \rangle \}$  is  $\text{ifthen}$ ’s equivalent of the while statement. It repeatedly ‘executes’  $\langle \text{statement} \rangle$  while  $\langle \text{test} \rangle$  evaluates to true. The following example demonstrates some of the functionality of the  $\text{ifthen}$  package.

```

\usepackage{ifthen}
\newcounter{counter}
\setcounter{counter}{5}

\begin{document}
  \[
    \thecounter = 0
    \whiledo
      {\not\(\thecounter=0\)}%
      {+1\addtocounter{counter}{-1}}\, .
  \]
\end{document}

```

The resulting output is ‘ $5 = 0 + 1 + 1 + 1 + 1 + 1.$ ’.

## 12.3 The calc Package

The `calc` package extends  $\text{\TeX}$  and  $\text{\LaTeX}$ ’s arithmetic. The `calc` package redefines the commands `\setcounter`, `\addtocounter`, `\setlength`, and `\addtolength`. As a result, these commands now accept infix expressions in their arguments. In addition the package provides useful commands such as `\widthof{⟨stuff⟩}`, `\ratio{⟨dividend⟩}{⟨divisor⟩}`, and so on, which don’t have a  $\text{\LaTeX}$  equivalent. The interested reader is referred to the package’s excellent documentation [Krab Thorub, Jensen and Rowley, 2005].

## 12.4 Looping

The  $\text{\LaTeX}$  kernel provides two kinds of `for` statements.

`\@for \var:=⟨list⟩\do \command`

Here `⟨list⟩` is a comma-delimited list. The items in `⟨list⟩` are bound to `\var` from left to right. After each binding, the command `\command` is carried out. (Of course, `\command` can also be a group.) As a simple example, ‘`\@for \var:=1,two\do{(\var)}`’ gives ‘`(1)(two)`’. Notice that it is imperative that the symbol ‘@’ is allowed in command sequence names. (The example in Figure 10.3 explains how to enable this.)  $\square$

`\@tfor \var:=⟨list⟩\do \command`

This is the ‘token’ version of the `\@for` command. In this case `⟨list⟩` is a list of tokens. The tokens in `⟨list⟩` are bound to `\var` from left to right. After each binding, the command `\command` is carried out. The following  $\text{\LaTeX}$  input gives us ‘`12332`’.

```

\def\swop#1#2{#2#1}
\@tfor \var:=1\swop\do{\var23}

```

$\square$

**Figure 12.1**

A tail recursion-based implementation of a lisp-like `\apply` command.

---

```

\documentclass[12pt]{article}

% \apply\cmd items\endApply:
% applies \cmd to each item in items, so
% \apply\twice a{bc}\endApply gives aabcbcb.
\def\apply#1{%
  \def\Apply##1{%
    \ifx##1\endApply%
      % The current argument is \endApply.
      % The following substitutes \fi for
      % the current substitution text of \Apply,
      % i.e. all tokens up to \Apply.
      \breakApply%
    \fi%
    #1{##1}% Apply \cmd to next item.
    \Apply% Tail recursive call.
  }%
  \Apply%
}
\def\breakApply#1\Apply{\fi}%
\def\twice#1{#1#1}

\begin{document}
  \apply\twice a{bc}d\endApply
\end{document}

```

---

## 12.5 Tail Recursion

This section studies tail recursion and demonstrates how it may be implemented using low-level T<sub>E</sub>X delimited commands. By carefully studying this section the interested reader should fully appreciate T<sub>E</sub>X and E<sub>T</sub><sub>E</sub>X programming in the large. The evaluation of the program which is depicted in Figure 12.1, demonstrates T<sub>E</sub>X expansion in its full glory. The program is based on [Fine, 1992]. There is one new ingredient in the example, which is related to decision making. For the purpose of this example, the construct `\ifx⟨A⟩⟨B⟩⟨statement⟩\fi` results in `⟨statement⟩` if the tokens `⟨A⟩` and `⟨B⟩` are equal. The key to understanding the example is observing that (1) `\breakApply` is applied only once inside `\Apply`, (2) that it is only applied when the token `\endapply` has been detected, and (3) that `\breakApply` gobbles the tokens which are following `\breakApply` in the substitution text of `\Apply`. The rest all boils down to tail recursion. It is left to the reader to determine the resulting output.



# CHAPTER 13

## User-defined Styles and Classes

13.1 User-defined Style Files

13.2 User-defined Class Files



**Part VI**  
**Miscellany**



# CHAPTER 14

## Beamer Presentations

THIS CHAPTER is an introduction to the `beamer` class, which is widely used for presenting computer presentations. Some people call such presentations *powerpoint presentations*. The class is seamlessly integrated with the `tikz` package, and lets you present *incremental* presentations, which are presentations which incrementally add material — text and graphics — to a page of the presentation.

The purpose of this chapter is *not* to explain all the possibilities of the `beamer` class but to explain just enough for what is needed for one or two presentations. The interested reader is referred to the excellent documentation [Tantau, Wright and Miletić, 2010] for further information.

The remainder of this chapter is as follows. In Section 14.1 we shall study *frames*, which correspond to one or several incremental slides on the screen. Section 14.2 explains the concept of *modal* presentations, which let you generate one or several versions of your presentation. For example, an in-class presentation and a set of lecture notes. This is continued by Section 14.3, which studies incremental presentations. Section 14.4 shows how to add some visual “alert” effects. This may be useful to highlight certain parts of the presentation. This chapter concludes with Section 14.5, which spends a few words on how you may personalise your presentations by adding a dash of style.

### 14.1 Frames

This section explains the frame environment, which is to a computer presentation what a page is to an article, a report, or a book. However, a frame may also be decorated with a frame title and a frame subtitle. Throughout this section we shall not worry about the overall look and feel of the presentation.

```
\begin{frame}[options] <frame material> \end{frame}
```

This is a simplified presentation of the frame environment (Section 14.2 provides a more complete description). When the output document is a computer presentation the `<frame material>` is turned into one or several slides in the output. Otherwise, it may result in one or several lines of text in the text of your output document.

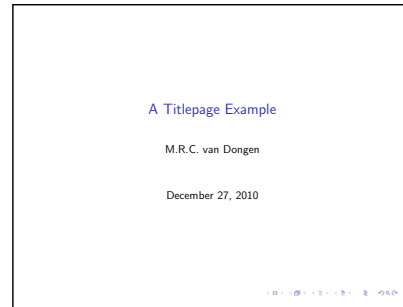
**Figure 14.1**

Creating a titlepage with the `beamer` class. The outline of the output slide is drawn for clarity. The little pictures in the lower right corner of the output are for navigation purposes.

```
\documentclass{beamer}

\title{A Titlepage Example}
\author{M.R.C. van Dongen}
\date{December 27, 2010}

\begin{document}
\begin{frame}[fragile]
\maketitle
\end{frame}
\end{document}
```



If the option ‘`fragile`’ is included in  $\langle \text{options} \rangle$ , then  $\langle \text{frame material} \rangle$  may contain any  $\text{\LaTeX}$  material. Including the option `fragile` is by far the easier: just use it. Omitting the option may result in errors. Tantau, Wright and Miletic [2010, Chapter 8] provides further information about the `fragile` option.

*The following is important:* the `\begin{frame}` and `\end{frame}` commands should be on a line of their own and there should be no spaces before the `\begin` and `\end`. □

```
\frametitle{ $\langle \text{frame title} \rangle$ }
```

This defines a frame title, which is usually typeset at the top of the resulting slides of a computer presentation. The frame title is only included if the output document is text-based. However, as we shall see in Section 14.2 it is possible to turn the frame title off for such documents. Turning the frame title off is also possible by redefining the `\frametitle` command. □

```
\framesubtitle{ $\langle \text{frame subtitle} \rangle$ }
```

This defines a subtitle for the frame. The subtitle is usually typeset below the frame title. □

Figure 14.1 provides the first `beamer` example. As you may see from the example, it looks like a regular  $\text{\LaTeX}$  document with a `\title`, `\author`, and `\maketitle` command. However, since `beamer` is a document class, its name is included in the `\documentclass` argument. The command `\maketitle` is put in a frame

Figure 14.2 demonstrates a simple `beamer` frame. The frame has a frame title and subtitle and its body consists of an itemised list.

The `beamer` class is nice when it works but it may lead to some unexpected complications. For example, defining an environment like the following may not work.

```
\newenvironment{myframe}[0]
{ \begin{frame}[fragile] }
{ \end{frame} }
```

Don't Try this at Home

Explaining why this environment doesn't work is beyond the scope of this chapter. As a general rule, automating `beamer` commands may

**Figure 14.2**  
Creating frame titles. The  
outline of the output slide is  
drawn for clarity.

```
\begin{frame}[fragile]
  \frametitle{A Slide}
  \framesubtitle{An Example}

  \begin{itemize}
    \item Hello world.
    \item Bonjour monde.
  \end{itemize}
\end{frame}
```



not always work: don't try it unless you have time. The manual [Tantau, Wright and Miletic, 2010] is the ultimate source of information for what is and isn't possible.

## 14.2 Modal Presentations

This section shows how to exploits beamer's *modes*, which let you generate several kinds of output documents from the same source. Here different output documents are not only allowed to have a different style of presentation but also different content. The following are beamer's basic modes.

**beamer** This is the default mode, which is what “beamer” is in. It corresponds to a computer presentation with one or several slides per frame.

**second** This mode is for outputting material to a second output screen.

**handout** This mode is for handouts. If a frame in the input is typeset in this mode, then it results in one output slide. This is different from the default mode, where one input frame may result in several output slides.

**trans** This mode is for creating transparencies. Having such an option almost seems like an anachronism. However, having a presentation in the form of transparencies may be useful as a backup resource, e.g., when presenting away from home.

**article** This mode is for typesetting text using a different existing  $\text{\LaTeX}$  class. For example, this book was typeset using  $\text{\LaTeX}$ 's `book` class in beamer's `article` mode. Doing this requires a slightly different approach. This time, you use the `\documentclass` to load the different class and use the `\usepackage` command to import the `beamerarticle` package. Figure 14.3 demonstrates how to do this. In this example, all frame titles and frame subtitles are turned off.

The `beamer` class is always in one of these five modes. By providing the mode as an optional argument to the class you determine the mode. If

**Figure 14.3**

Using the `beamerarticle` package.

---

```
\documentclass{book}
\usepackage{beamerarticle}
\makeatletter
\def\frametitle{%
  \@ifnextchar<%
    {\@frametitle@lt}%
    {\@frametitle@lt<>}%
}
\def\@frametitle@lt<#1>#2{}
\makeatother
```

---

**Figure 14.4**

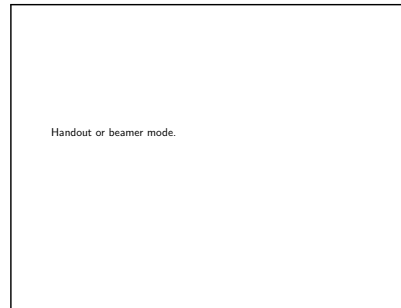
Using modes. The outline of the slide is drawn for clarity.

---

```
\documentclass[handout]
{beamer}

\begin{document}
\begin{frame}
  <handout|beamer>
  {fragile}
  Beamer or handout mode.
\end{frame}
\begin{frame}
  <beamer>
  {fragile}
  Beamer mode.
\end{frame}
\end{document}
```

---



you omit the mode then beamer will be in, well, beamer mode.

In addition, beamer has the following auxiliary modes:

**all** This is for all modes.

**presentation** This mode is for all “presentation” modes, so all modes except for article.

Having defined beamer’s modes, it’s time to revisit to its frame environment.

```
\begin{frame}<<overlay specification>[<options>]<frame material>\end{frame}
```

Here ‘<overlay specification>’ works like an optional argument. Modes in <overlay specification> determine whether the frame should be typeset. For example, if <overlay specification> is equal to ‘article’ and beamer is in ‘beamer’ mode then the frame is not typeset. You may combine modes using ‘|’ as a separator: ‘beamer|handout’. □

Figure 14.4 demonstrates the basic mode mechanism. There are two frames. The first frame is typeset in handout or beamer mode. The second frame is only typeset in beamer mode. The beamer class is started in handout mode. This explains why only the first frame is typeset.



Other beamer commands and environment may also accept overlay specifications. Having to specify the same overlay specification is tedious and prone to errors. The following commands help avoiding redundant overlay specifications.

`\mode<mode specification>{<text>}`

This results in inserting `<text>` if beamer's mode corresponds to `<mode specification>`. Note that this only works if the first non-space character following the `'>'` is an opening brace. □

`\mode<mode specification>`

This filters subsequent text which does not correspond to `<mode specification>`. Note that this only works if the first non-space character following the `'>'` is *not* a brace. □

`\mode*`

If beamer is in presentation mode, then this command causes beamer to ignore text outside frames. If beamer is in article mode, then this command has no effect. □

## 14.3 Incremental Presentations

*Incremental presentations* incrementally unveil parts of the content of a frame environment. Typically, this is done by displaying the next item in an itemised list. The beamer class also provides annotations which let you present material on the  $n^{\text{th}}$  output slide which is generated from a given frame. The following are some of the relevant commands. More information may be found in [Tantau, Wright and Miletic, 2010, Chapter 9].

`\pause`

This inserts a pause stop at the corresponding position. The net effect of it is that this increases the number of output slides which are generated from the current frame. The pause stop separates the slides before and after the position of the `\pause` command. □

`\pause[<number>]`

This command unveils the text following the `\pause` command from Slide `<number>` and onwards. □

Figure 14.5 provides an example of the `\pause` command. The frame in the input results in three slides in the output. The first slide contains the first item of the itemised list. The second slide contains the first, the second, and the third item. The last slide contains all items of the itemised list. It is assumed that beamer is in beamer mode.

The beamer class redefines the standard `\item` command. This version of the command results in displaying the corresponding item on the slides corresponding to an *overlay specification*, which defines the slides on which the item is displayed.

`\item<overlay specification>`

The corresponding item is typeset on the slides corresponding to `<overlay specification>`. On the remaining slides, the item is typeset in invisible ink. Possible `<overlay specification>`s are as follows:

**Figure 14.5**

Using the `\pause` command. The `frame` environment results in three output slides, the second of which is shown to the right. The outline of the slide is drawn for clarity.

```
\begin{frame}[fragile]
\begin{itemize}
\item First. \pause
\item Second.
\item Third. \pause
\item Last.
\end{itemize}
```

**Figure 14.6**

Using overlay specifications. The `frame` environment results in three output slides, the second of which is shown to the right. The outline of the slide is drawn for clarity.

```
\begin{frame}[fragile]
\begin{itemize}
\item<1-2> First.
\item<3,4> Second.
\item<2> Third.
\item Last.
\end{itemize}
```



`<number>` This corresponds to Slide `<number>`.

`<number>-` This corresponds to Slide `<number>` and further.

`-<number>` This corresponds to Slides `1-<number>`.

`<number>_1-<number>_2` This corresponds to Slides `<number>_1-<number>_2`.

`<overlay specification>_1,<overlay specification>_2` This combines `<overlay specification>_1` and `<overlay specification>_2`.

Other commands may also accept overlay specifications. The reader is referred to the class documentation [Tantau, Wright and Miletic, 2010] for further information.  $\square$

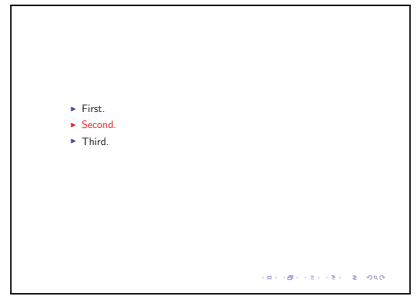
**Intermezzo.** *The beamer class defines many more commands which may be useful when creating incremental presentations. Incremental presentations may look slick, but creating them takes precious time. Peyton Jones, Huges and Launchberry [1993] argue that some of your audience may not even like incremental presentations which unveil an itemised list one item at a time. It is the content of the presentation which determines the quality — not the visual effects. If you're a student then it is not likely that you will have to give many presentations in your life as a student. Therefore, you should consider doing yourself and your audience a favour: minimise the visual effects and spend the time you save on the content of the presentation.*

The `tikz` package and the `beamer` class are seamlessly integrated. This means you can also create incremental presentations with `tikz` pictures. Such presentations may be highly effective. However, creating them may take a *lot* of time ....

**Figure 14.7**

Adding visual alerts. The frame to the left results in four slides. The output of the third slide is depicted to the right. The outline of the slide is drawn for clarity.

```
\begin{frame}[fragile]
\begin{itemize}
\item<alert@2> First.
\item<alert@3> Second.
\item<alert@4> Third.
\end{itemize}
\end{frame}
```



## 14.4 Visual Alerts

A visual alert in a presentation uses colour to emphasise text. Using visual alerts is useful if you want to emphasise different parts of a frame at different times. It is especially useful if you're discussing items in a list and if you want to indicate which item is currently being discussed. The following are some related commands.

`\alert<overlay specification>{<text>}`

This emphasises `<text>` on the slides corresponding to `<overlay specification>`. Omitting `<overlay specification>` results in highlighting `<text>` on all slides. ☐

`\item<alert@<overlay specification>>`

This emphasises the corresponding item in a list on the slides corresponding to `<overlay specification>`. ☐

`\item<overlay specification>_1|alert@<overlay specification>_2>`

This results in displaying the item on the slides corresponding to `<overlay specification>_1` and emphasising the item on the slides corresponding to `<overlay specification>_2`. ☐

Figure 14.7 presents an example which uses visual alerts to highlight the different items in an itemised list.

## 14.5 Adding Some Style

The presentation to beamer has been quite minimal. The main reason for this is that learning to use the class takes time. For students — the main target audience — it is better if they stick to simple presentations and spend their time on the content of their presentation.

Having made these observations, it is good to note that some presentations benefit from some additional decoration. For example, a menu listing the sections in the presentation may help the audience recognise the structure of the presentation.

A beamer *theme* determines a certain aspect of the visual presentation. Currently, there are five beamer themes: presentation, colour, font, inner, and outer. The presentation themes are the easier ones to use because they define *everything* in the presentation. New beamer users are better off starting with a presentation theme because then they don't have worry about the presentation style. Most presentation themes are actually

**Figure 14.8**

Using a beamer theme. The  $\text{\LaTeX}$  input is a template which is used to demonstrate the effect of the different beamer themes in the remainder of this section. The outputs are obtained by substituting the name of the themes for  $\langle\text{theme}\rangle$  in the input.

---

```

\documentclass{beamer}
\usetheme[ $\langle\text{options}\rangle$ ]{ $\langle\text{theme}\rangle$ }

\title{Prime Number Presentation}
\institute{University of Alexandria}
\author{Euclid}

\begin{document}
\begin{frame}[fragile]
\maketitle
\end{frame}

\section{Main Result}

\begin{frame}[fragile]
\frametitle{There is No Largest Prime Number}
\framesubtitle{The Proof Uses  $\emph{Reductio ad Absurdum}$ }
\begin{theorem}
\begin{enumerate}
\item $\langle\text{alert@2}\rangle$  Suppose  $p$  were the largest prime number.
\item $\langle\text{alert@2}\rangle$  Let  $P$  be
            the product of the first  $p$  primes.
\item $\langle\text{alert@4}\rangle$  Then  $P + 1$  is not
            divisible by any prime.
\item $\langle\text{alert@5}\rangle$  Therefore,  $P + 1$  is also a prime.
            \qedhere
\end{enumerate}
\end{theorem}
\end{frame}

\section{Conclusion}

\begin{frame}[fragile]
The end.
\end{frame}
\end{document}

```

---

quite good. Seasoned beamer users may want to spend some time on fine-tuning their own style.

The remainder of this section presents four presentation themes which seem nice for a first presentation with only a few slides. More information about themes may be found in the documentation [Tantau, Wright and Milić, 2010].

Figure 14.8 demonstrates the input which was used to demonstrate the different themes. The input is inspired by the beamer documentation. The resulting outputs are listed in Figures 14.9–14.12. For each theme,

the figure contains the fifth slide, i.e., the fourth slide of the second frame.

Figure 14.9

Sample output of beamer's default theme. The outline of the slide is drawn for clarity.

# There is No Largest Prime Number

The Proof Uses *Reductio ad Absurdum*

## Theorem

1. Suppose  $p$  were the largest prime number.
2. Let  $\Pi$  be the product of the first  $p$  primes.
3. Then  $\Pi + 1$  is not divisible by any prime.
4. Therefore,  $\Pi + 1$  is also a prime.





Figure 14.10

Sample output of beamer's Boadilla theme. The option `secheader` was passed as an option to the `\usetheme` command. The outline of the slide is drawn for clarity.

Main Result

# There is No Largest Prime Number

The Proof Uses *Reductio ad Absurdum*

Theorem

➊ Suppose  $p$  were the largest prime number.

➋ Let  $\Pi$  be the product of the first  $p$  primes.

➌ Then  $\Pi + 1$  is not divisible by any prime.

➍ Therefore,  $\Pi + 1$  is also a prime.

□

Euclid (University of Alexandria)

Prime Number Presentation

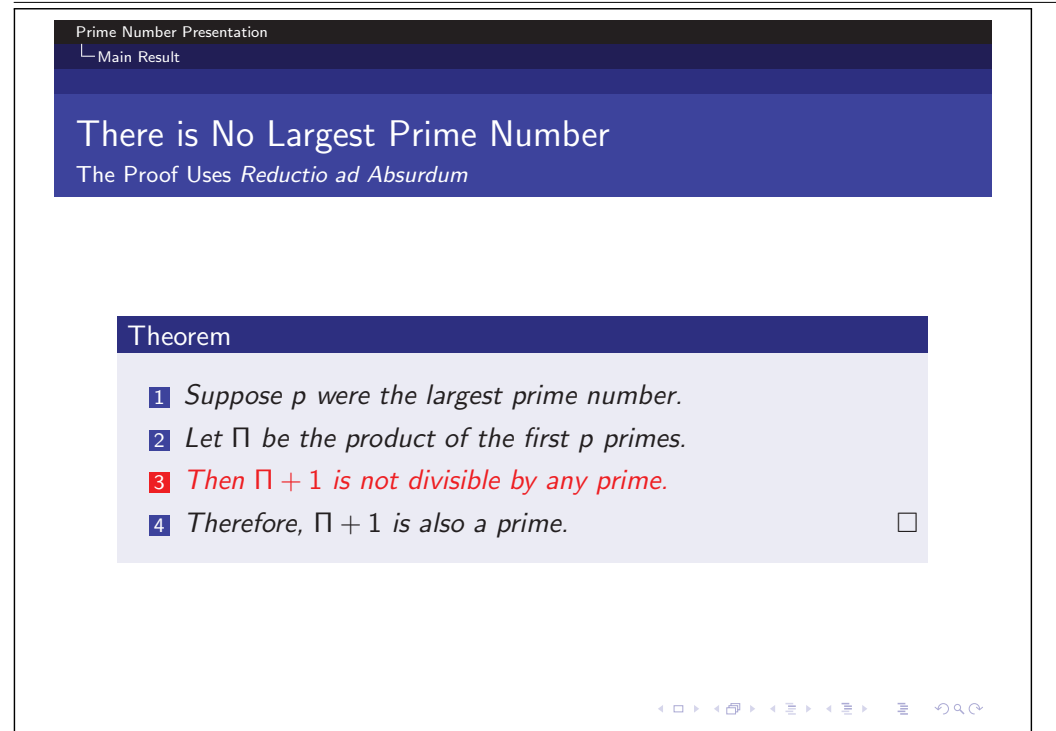
December 28, 2010 2 / 2

Figure 14.9 depicts the sample output of `beamer`'s default theme. This theme is very sober and implements visual alerts by typesetting text in red, which is the default for visual alerts.

The `Boadilla` theme, which is depicted in Figure 14.10, is a bit more lively. Using this theme also adds some information about the “author” at the bottom of each slide. Passing the option `secheader` also lists the current section and subsection at the bottom of the slides.

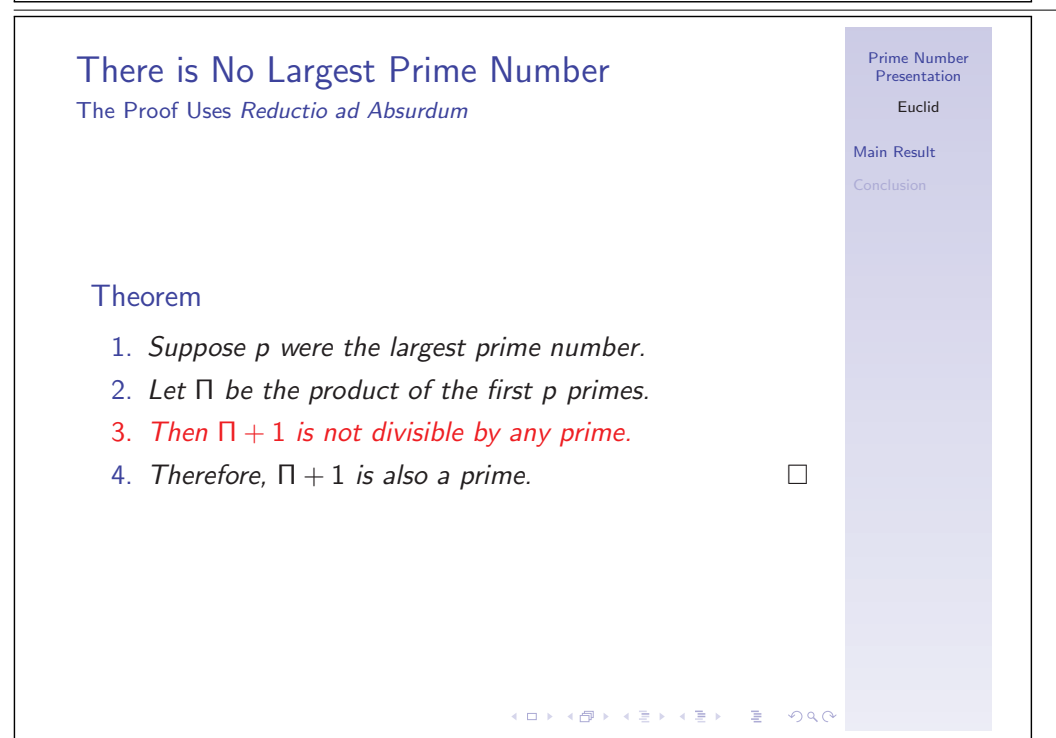
**Figure 14.11**

Sample output of beamer’s `Antibes` theme. The outline of the slide is drawn for clarity.



**Figure 14.12**

Sample output of beamer’s `Goettingen` theme. The outline of the slide is drawn for clarity.



Sample output of the `Antibes` theme is depicted in Figure 14.11. On top of the information which is provided by `Boadilla`, this theme also

provides a tree-line navigation menu at the top. This kind of information may be useful for the audience because it helps them recognise the presentation structure and helps them determine where you “are” in the presentation.

The final theme is *Goettingen*. It is depicted in Figure 14.12. This theme is for long presentations and comes equipped with a sidebar containing a table of contents. This theme accepts the following options.

**hideallsubsections** This removes subsection information from the sidebar.

**hideothersubsections** With this option only the subsections of the current section are shown in the sidebar.

**left** This puts the sidebar to the left. This is the default behaviour.

**right** This puts the sidebar on the right.

**width=⟨dimension⟩** This sets the width of the sidebar. Providing a width of zero hides the sidebar.





# CHAPTER 15

## Installing L<sup>A</sup>T<sub>E</sub>X and Friends

THIS CHAPTER DESCRIBES how to install a widely available L<sup>A</sup>T<sub>E</sub>X distribution called T<sub>E</sub>X Live, how to configure T<sub>E</sub>X Live, how to install L<sup>A</sup>T<sub>E</sub>X class and style files, and how to install and use new fonts. The remainder of this chapter is as follows. Section 15.1 explains how to install the T<sub>E</sub>X Live distribution. This is followed by Section 15.2, which explains how to configure the T<sub>E</sub>X Live distribution. Section 15.3 explains how to install new classes and packages (style files). Section 15.4 explains how to install L<sup>A</sup>T<sub>E</sub>X fonts — this is not an easy task. Section 15.5 describes the easier task of installing Unix .otf and .ttf fonts. In Section 15.6 it is shown how these Unix fonts can be used directly in L<sup>A</sup>T<sub>E</sub>X using the fontspec package. Section 15.7 concludes this chapter by providing some clues on how to install T<sub>E</sub>X Live with a package manager such as apt-get.

### 15.1 Installing T<sub>E</sub>X Live

One of the easier L<sup>A</sup>T<sub>E</sub>X distribution to install is T<sub>E</sub>X Live. T<sub>E</sub>X Live may be downloaded from the *CTAN!* (CTAN!) from `ftp.heanet.ie/pub/CTAN/tex/systems/texlive/Images/`.

The T<sub>E</sub>X Live distribution comes as an .iso image. Installing it is child's play (throughout it is assumed you have root access).

The following demonstrates how to install the distribution. In the example, it is assumed that you've downloaded the .iso image and that it is called `texlive.iso`. If you create a Compact Disk (CD) with the image then you can run the command `./install-tl.sh` from the CD's root directory. However, there is no need to create a CD: you can directly mount the .iso image. The following show how this is done.

We start by creating a directory `/mnt/texlive` and by mounting the .iso image using `mount` and the `loop` option.

```
# mkdir /mnt/texlive
# mount -t iso9660 -o loop texlive.iso /mnt/texlive
```

Unix Session

The image being mounted, we continue by going to the mounted directory and by running the install program, which is called `install-tl.sh`:

```
# cd /mnt/texlive
# ./install-tl.sh
```

Unix Session

The install program is interactive and is pretty intuitive. Once you've selected your configuration setting — the default setting should do — you select Option I in the main menu and the installation begins. After the installation you unmount and remove the directory `/mnt/texlive`. The following shows how to do this.

```
# cd /
# umount /mnt/texlive
# rmdir /mnt/texlive
```

Unix Session

## 15.2 Configuring T<sub>E</sub>X Live

Having installed the T<sub>E</sub>X Live source files we're almost done with our installation. All that remains is related to configuration. This section gives some minimal clues on how to configure L<sup>A</sup>T<sub>E</sub>X and friends. Throughout L<sup>A</sup>T<sub>E</sub>X is used as a shorthand for L<sup>A</sup>T<sub>E</sub>X and friends. In the following, Section 15.2.1 explains how to configure the Unix search path and Section 15.2.2 explains how to configure the L<sup>A</sup>T<sub>E</sub>X search path.

### 15.2.1 Adjusting the PATH

First we adjust the `PATH` environment variable, which is needed by Unix to locate your executables. We configure the `PATH` variable by adding the path name of the directory containing the L<sup>A</sup>T<sub>E</sub>X executables. The following example shows how to do this. In the example, it is assumed that the directory containing the executables is `/usr/local/texlive/2007/bin/i386-linux`.

```
~> PATH=/usr/local/texlive/2007/bin/i386-linux:${PATH}
~> export PATH
```

Unix Session

The best thing is putting these commands in your `.login` file.

### 15.2.2 Configuring TEXINPUTS

The final task of our configuration chores is setting up the L<sup>A</sup>T<sub>E</sub>X environment variable `TEXINPUTS`, which is used to specify the search path for input files. This variable defines a sequence of paths which are searched by L<sup>A</sup>T<sub>E</sub>X when it is looking for input files.

As with the `PATH` environment variable, the paths are separated with colon (:) characters. When looking for an input file called `file`, the paths in `${TEXINPUTS}` are searched from left to right. If one of them, path say, contains `file`, and if path is the first such path, then L<sup>A</sup>T<sub>E</sub>X will assume that path is the directory from which it should load `file`. Otherwise L<sup>A</sup>T<sub>E</sub>X will try and locate `file` in its default directories. This mechanism allows you to help L<sup>A</sup>T<sub>E</sub>X locate files which are located in non-standard locations and override the default locations. It is especially

useful for setting up a local directory with user-specific class and style files.

The `TEXINPUTS` mechanism is more flexible than the `PATH` mechanism. By adding a double forward slash (`//`) to the end of a path, L<sup>A</sup>T<sub>E</sub>X will search the path recursively. The following is a typical configuration, which tells L<sup>A</sup>T<sub>E</sub>X to first search the current directory, next recursively search the directory `${HOME}/LaTeX/styles`, and finally recursively search the directory `${HOME}/LaTeX/mpost`.

```
~> export LaTeX=${HOME}/LaTeX
~> export TEXINPUTS=.:${LaTeX}/styles//:${LaTeX}/mpost//:
```

Unix Session

## 15.3 Installing Classes and Packages

This section explains how to install user-specific class and package (style) files which are not part of the standard distribution. The mechanism for installing these files as it is presented here allows the users to use their own versions of class and style files (as opposed to other versions which are installed in the main installation).

To install user-specific class and style files, it is strongly recommended this be done in a special-purpose directory which is owned by the user. The sole purpose of the directory should be to store class and package files and other input files which are used frequently as input for other source files. By properly configuring the `TEXINPUTS` variable — this is explained in Section 15.2.2 — the user can force L<sup>A</sup>T<sub>E</sub>X to first recursively search the special-purpose directory for their own input files. This effectively allows them to install and use more recent (or older) versions of class and style files as well as install their own user-specific files in a location where L<sup>A</sup>T<sub>E</sub>X will find them.

Let's assume we want to install a new style or class file. To install the file we do the following.

**Download files** Download the files. If needed uncompress them. It is good practice to put the files in a separate directory in your special-purpose directory. This makes it easy to locate the package-related files and uninstall them.

**Extract files** Run L<sup>A</sup>T<sub>E</sub>X on the `.ins` file.

**Create documentation** Run L<sup>A</sup>T<sub>E</sub>X on the `.dtx` file. You may have to do this more than once to get cross-references right. Likewise, you may have to create index files if `.idx` files are created as a result of the compilation process. Section 1.8.4 describes how to do this.

**Update package database** Run the `texhash` program. This adds the location of the files to the package database, which allows L<sup>A</sup>T<sub>E</sub>X to find your files on subsequent runs.

## 15.4 Installing L<sup>A</sup>T<sub>E</sub>X Fonts

This section briefly explains how to install new fonts. **To Do.**

## 15.5 Installing Unix Fonts

If you haven't done it before then installing fonts the L<sup>A</sup>T<sub>E</sub>X way may a lot of work. However, with the arrival of the beautiful `fontspec` package you can now directly use any Unix `.ttf` or `.otf` font.<sup>1</sup> This reduces the task of using non-standard fonts to the installation of Unix fonts. In the following we shall install fonts globally. To do this you need root permission.

To explain the mechanism, we shall study how to install the Inconsolata monospaced font. You may download the font from <http://www.levien.com/type/myfonts/inconsolata.html>.

- To keep the management of your fonts under control, it is recommended that you put your `.otf` and `.ttf` files in a special directory for each specific font. In the following it is assumed all such directories are located in `${HOME}/.fonts`.
- Since we decided to have a special directory for each font, our next step is to create a directory called `Inconsolata` in the directory `${HOME}/.fonts`.
- We continue by downloading the file `Inconsolata.otf` and save it in the new directory.
- Now that we've saved the font, we have to make sure we can use it. To do this we have to build the font information cache files. Building these files may be done with the `fc-cache` program. Our decision to install *all* our fonts in the directory called `${HOME}/.fonts` makes the installation very easy and easy to automate. In the following example, we run `fc-cache` recursively on our directory `${HOME}/.fonts` and make it (really) force the installation. As may be noticed from the example, the program is run in the user's home directory.

```
~> su Unix Session
Password:
# fc-cache -fvr ./fonts
```

## 15.6 Using the fontspec Package

The `fontspec` package provides an easy mechanism for configuring fonts. It significantly reduces the task of installing fonts. The `fontspec` package allows users of either `xetex` or `luatex` to load OpenType fonts in a L<sup>A</sup>T<sub>E</sub>X document. The package more than likely comes with your L<sup>A</sup>T<sub>E</sub>X distribution but can also be downloaded from CTAN.

<sup>1</sup> Some of the features of the `fontspec` package are described in Section 15.6.

**Figure 15.1**

Using the fontspec package.

---

```
\usepackage{fontspec}
% Without the following, things may not work the LaTeX way
\defaultfontfeatures{Mapping=tex-text}

\setsansfont[Ligatures=Rare,Numbers={SlashedZero}]{Arial}
\setromanfont[Ligatures=Rare,Numbers={OldStyle}]{Garamond}
\setmonofont{Inconsolata}
```

---

It is impossible to explain the functionality of the fontspec package in full detail. Figure 15.1 provides a minimal example which shows how the commands `\setsansfont`, `\setromanfont`, and `\setmonofont` may be used to use other non-standard fonts. As is suggested by the names, the commands are for defining the default sans serif font, the default roman (serif) font, and the default mono-space font. It is also possible to use fontspec in combination with locally installed fonts. The reader is referred to the comprehensive fontspec documentation [Robertson, 2008] for further information about the beautiful package.

## 15.7 Package Managers

With the advance of package managers, such as `apt-get` (Debian, Ubuntu, ...) installing L<sup>A</sup>T<sub>E</sub>X and friends has become much easier. However, a possible disadvantage is that it may not always be possible to get the most recent version of T<sub>E</sub>X Live.

Installing T<sub>E</sub>X Live with `apt-get` requires the following two commands and the typing of the root password:

```
~> sudo apt-get update
[sudo] password for user:
~> sudo apt-get install texlive
```

Unix Session

A full version of T<sub>E</sub>X Live may be installed as follows:

```
~> sudo apt-get install texlive-full
```

Unix Session

An additional advantage of installing L<sup>A</sup>T<sub>E</sub>X with a package manager such as `apt-get` is that there is no need to adjust the `PATH` environment variable. If you don't require any user-specific options, then configuring the environment variable `TEXINPUTS` may not be necessary either.



# CHAPTER 16

## Resources

THIS CHAPTER PROVIDES some pointers to useful resources. Most of them are available online.

### 16.1 Books about T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X

The following are great books about T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X. Many of them are freely available.

- [Knuth, 1990]: The original reference to T<sub>E</sub>X. The source code is freely available.
- [Lamport, 1994]: From the creator of L<sup>A</sup>T<sub>E</sub>X.
- [Abrahams, Hargreaves and Berry, 2003]: Good book about T<sub>E</sub>X. Freely available.
- [Eijkhout, 2007]: Good book about T<sub>E</sub>X. Freely available.
- [Goossens, Mittelbach and Samarin, 1994]: L<sup>A</sup>T<sub>E</sub>X book.
- [Goossens, Rahtz and Mittelbach, 1997]: Graphics and L<sup>A</sup>T<sub>E</sub>X.
- [Goossens, Rahtz and Mittelbach, 1999]: Creating navigable documents with T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X.

### 16.2 Bibliography Resources

Detailed information about managing your bibliography with L<sup>A</sup>T<sub>E</sub>X is [Fenn, 2006]. More information may be found at <http://en.wikipedia.org/wiki/BibTeX>.

### 16.3 Articles by the L<sup>A</sup>T<sub>E</sub>X<sub>3</sub> Team

Articles by the L<sup>A</sup>T<sub>E</sub>X<sub>3</sub> Team. All are available online.

- [The L<sup>A</sup>T<sub>E</sub>X<sub>3</sub> Project, 2001b]: L<sup>A</sup>T<sub>E</sub>X User guide.
- [The L<sup>A</sup>T<sub>E</sub>X<sub>3</sub> Project, 2001c]: Explains how to create L<sup>A</sup>T<sub>E</sub>X class and style files. A seemingly very related etoolbox package.
- [The L<sup>A</sup>T<sub>E</sub>X<sub>3</sub> Project, 2001a]: Explains the L<sup>A</sup>T<sub>E</sub>X Font Selection Mechanism.
- [The L<sup>A</sup>T<sub>E</sub>X<sub>3</sub> Project, 2001]: Explains how to configure L<sup>A</sup>T<sub>E</sub>X.
- [Braams et al., 2001]: Describes the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> sources.

## 16.4 L<sup>A</sup>T<sub>E</sub>X Articles, Course Notes and Tutorials

L<sup>A</sup>T<sub>E</sub>X and T<sub>E</sub>X articles, course notes, and tutorials: All are available online.

- [Oetiker et al., 2007]: Most-cited L<sup>A</sup>T<sub>E</sub>X tutorial.
- [Voß, 2009]: Comprehensive and interesting presentation of commands and environments in math mode (including  $\mathcal{AMS}$ -L<sup>A</sup>T<sub>E</sub>X-provided symbols).
- [Eijkhout, 2004]: Course about the computer science behind L<sup>A</sup>T<sub>E</sub>X.
- [Krishnan, 2003]: A very comprehensive L<sup>A</sup>T<sub>E</sub>X tutorial.
- [Heck, 2005a]: Hands-on L<sup>A</sup>T<sub>E</sub>X tutorial.
- [Reckdahl, 2006]: Explains how to include imported graphics in L<sup>A</sup>T<sub>E</sub>X and pdfL<sup>A</sup>T<sub>E</sub>X.
- [Pakin, 2005]: A list of symbols and how to produce them with L<sup>A</sup>T<sub>E</sub>X.
- [Pakin, 2006]: A Frequently Asked Questions (FAQ). Click on the sensitive areas and you'll be shown how to produce it.
- [Maltby, 1992]: A T<sub>E</sub>X tutorial.

## 16.5 METAPOST Articles and Tutorials

The following are some interesting METAPOST tutorials. All are available online.

- [Hagen, 2002]: METAPOST tutorial in the form of small examples.
- [*Making MetaPost Outlines*]: Describes how to turn L<sup>A</sup>T<sub>E</sub>X into METAPOST graphics.
- [Hurlin, 2007]: Practical introduction to METAPOST.
- [Heck, 2005b]: Hands-on introduction to METAPOST.

## 16.6 On-line Resources

- Comprehensive T<sub>E</sub>X Archive Network (CTAN):
  - Home: <http://www.ctan.org>.
  - Search: <http://www.ucc.ie/cgi-bin/ctan>.
- T<sub>E</sub>X Users Group (TUG):
  - Home: <http://www.tug.org/>.
  - Resources: <http://www.tug.org/interest.html>.
- UK T<sub>E</sub>X FAQ: <http://www.tex.ac.uk/cgi-bin/texfaq2html>.
- The American Mathematical Society's T<sub>E</sub>X Pages: <http://www.ams.org/tex/>.
- Pages related to the tikz package:
  - Sourceforge page for the tikz package: <http://sourceforge.net/projects/pgf/>.



- Fauskes.net impressive list of examples: <http://www.texample.net/tikz/examples/>.
- Alain Matthes' beautiful tkz-berge package for drawing graphs: <http://www.altermundus.fr/pages/download.html>.
- Donald Knuth's homepage (creator of  $\text{\TeX}$ ): <http://www-cs-faculty.stanford.edu/~knuth/>.
- John Hobby's METAPOST Pages: <http://cm.bell-labs.com/who/hobby/MetaPost.html>.
- The  $\text{\LaTeX}$  Project: <http://www.latex-project.org/>.
- Pdf $\text{\TeX}$  support: <http://www.tug.org/applications/pdftex/>.
- Generating PDF with animations and  $\text{\LaTeX}$ : <http://darkwing.uoregon.edu/~noeckel/PDFmovie.html>.
- A list of METAPOST links may be found at <http://cswb.ucc.ie/~dongen/mpost/links.html>.
- The ghostview resource page: <http://pages.cs.wisc.edu/~ghost/gv/index.htm>.
- The gsvieview page: <http://pages.cs.wisc.edu/~ghost/gsvieview/>.
- The Auc $\text{\TeX}$  pages. Auc $\text{\TeX}$  is a  $\text{\LaTeX}$  editing environment, which includes real-time viewing of the final output in a separate window. The package may be downloaded from <http://www.gnu.org/software/auctex/>.
- The  $\text{\TeX}$ nicCenter pages.  $\text{\TeX}$ nicCenter is an integrated  $\text{\LaTeX}$  development environment (IDE). The package may be downloaded from <http://www.texniccenter.org/>.

## 16.7 YouTube Resources

The following are some YouTube resources.

- Using  $\text{\LaTeX}$  with MikTeX:
  - <http://sites.google.com/site/wdoerner/latex>;
  - [http://www.youtube.com/watch?v=mVq16Tl\\_W20&feature=related](http://www.youtube.com/watch?v=mVq16Tl_W20&feature=related).
- Using  $\text{\LaTeX}$  with Lyx: Parts 1–5:
  - <http://www.youtube.com/watch?v=m4cEAVmLegg&feature=related>;
  - <http://www.youtube.com/watch?v=Wq9ti7GGHrs&feature=related>;
  - <http://www.youtube.com/watch?v=05okEyYQ-0g&feature=related>;
  - <http://www.youtube.com/watch?v=sJpfyydhAzo&feature=related>;
  - [http://www.youtube.com/watch?v=wyV\\_cjV-c1I&feature=related](http://www.youtube.com/watch?v=wyV_cjV-c1I&feature=related).
- MikTeX and Texniccenter installation:

- [http://www.youtube.com/watch?v=NeNOj\\_Ulys8&feature=related](http://www.youtube.com/watch?v=NeNOj_Ulys8&feature=related).

## 16.8 English

- [Thomson and Martinet, 1986a]: A brilliant English Grammar! This book comes with two exercise books [Thomson and Martinet, 1986b; Thomson and Martinet, 1986c].
- [Allan, 2001]: Great punctuation reference guide with information about need for, origin of, and how to use punctuation.
- [Trask, 1997]: Punctuation reference guide.
- On-line Oxford English Dictionary: <http://dictionary.oed.com/>.

# **Part VII**

## **References and Bibliography**



# Indices

## Index of L<sup>A</sup>T<sub>E</sub>X and T<sub>E</sub>X Commands

- \', 42
- \(, 218, 219
- \), 218, 219
- \+, 56, 57
- \,, 44, 76, 117, 118, 120, 144–152, 154–158, 160, 161, 163, 165, 166, 170–174, 219
- \-, 56, 57
- \., 42
- \:, 174
- \;, 174, 181, 184
- \=, 42, 56, 57
- \>, 57
- \[, 145, 150–152, 154–158, 160–162, 165, 166, 170, 171, 173, 219
- \#, 40
- \\$ , 40
- \%, 40
- \&, 40
- \^, 42
- \\_ , 40
- \\, 47, 51, 56, 118, 120, 144–149, 151, 157, 185, 186
- \\\*, 47
- \{ , 40, 151
- \} , 40
- \~, 42
- \], 145, 150–152, 154–158, 160–162, 165, 166, 170, 171, 173, 219
- \', 42
- \, 45
- \AA, 43
- \aa, 43
- \abstract, 12
- \acute, 164
- \addcontentsline, 15
- \addlegendentry, 125, 128, 129, 131, 133
- \addlinespace, 54
- \addplot, 125–135
- \addplot+, 132
- \address, 34
- \addtocounter, 214, 219
- \addtolength, 217, 219
- \AE, 43
- \ae, 43
- \aleph, 178
- \alert, 231
- \Alph, 61, 62
- \alph, 61–64
- \alpha, 142, 143, 162
- \amalg, 175
- \AND, 218
- \and, 11, 218
- \angle, 178
- \appendix, 16
- \approx, 161, 176
- \approxeq, 176
- \arabic, 61, 62
- \arccos, 158
- \arcsin, 158
- \arctan, 158
- \arg, 158
- \arraycolsep, 56
- \arrayrulewidth, 56
- \arraystretch, 56
- \ast, 175
- \asympt, 176
- \author, 9–11, 226, 232
- \b, 42
- \backepsilon, 176
- \backmatter, 15
- \backsim, 176
- \backsimeq, 176
- \backslash, 39, 40, 154, 178
- \bar, 164
- \baselinekip, 216
- \because, 176
- \begin, 10–12, 15, 17, 20, 25, 29, 34, 47–51, 53–55, 57, 60–64, 70–72, 76–82, 84–93, 95, 98–100, 103, 104, 107–110, 118–

- 120, 125, 127, 129, 131,  
133, 134, 144–149, 151,  
157, 159, 160, 162, 163,  
165, 170, 172, 181, 183,  
184, 187, 196, 197, 200,  
203, 206, 214, 215, 218–  
220, 225–228, 230–232
- `\begingroup`, 42
- `\beta`, 32, 142, 143, 162
- `\between`, 176
- `\bfseries`, 41, 42, 49, 188
- `\bibliography`, 15, 24, 25, 28,  
29
- `\bibliographystyle`, 22, 25, 28
- `\bibname`, 15
- `\BibTeX`, 28
- `\bigcap`, 156
- `\bigcirc`, 175
- `\bigcup`, 156
- `\bigl`, 163
- `\bigodot`, 156
- `\bigoplus`, 156
- `\bigotimes`, 156
- `\bigr`, 163
- `\bigsqcup`, 156
- `\bigtriangledown`, 175
- `\bigtriangleup`, 175
- `\biguplus`, 156
- `\bigvee`, 156
- `\bigwedge`, 156
- `\binom`, 141, 144, 157
- `\bmod`, 159, 181
- `\boldmc`, 120
- `\boldsymbol`, 174
- `\boolean`, 218
- `\bot`, 178
- `\bottomrule`, 53, 54, 118, 120
- `\bowtie`, 176
- `\Box`, 178
- `\breve`, 164
- `\bullet`, 175
- `\Bumpeq`, 176
- `\bumpeq`, 176
- `\c`, 42
- `\cap`, 175
- `\caption`, 69–71, 118, 119, 180,  
181, 197
- `\Case`, 183
- `\cc`, 34
- `\cdot`, 171, 175
- `\cdotp`, 170, 171
- `\cdots`, 170, 171
- `\centering`, 72
- `\cfrac`, 154
- `\chapter`, 14–16, 18, 20, 30
- `\chapter*`, 14, 15
- `\check`, 164
- `\chi`, 143
- `\circ`, 175
- `\circeq`, 176
- `\cite`, 3, 22, 23, 25, 27, 28, 40,  
45, 155
- `\Citeauthor`, 27
- `\citeauthor`, 27
- `\citep`, 27
- `\Citet`, 27
- `\citet`, 27
- `\citeyear`, 27
- `\cleardoublepage`, 29
- `\clearpage`, 29
- `\cline`, 52, 56
- `\closing`, 34
- `\clubsuit`, 178
- `\cmidrule`, 54
- `\colon`, 166, 170, 171
- `\colorlet`, 84
- `\commandkey`, 211
- `\cong`, 176
- `\coprod`, 156
- `\cos`, 158
- `\cosh`, 158
- `\cot`, 158
- `\coth`, 158
- `\csc`, 158
- `\csname`, 202, 203, 206
- `\cup`, 175
- `\curlyeqprec`, 176
- `\curlyeqsucc`, 176
- `\d`, 42
- `\dagger`, 175
- `\dashv`, 176

- `\date`, 10, 11, 226
- `\ddagger`, 175
- `\ddddot`, 164
- `\dddot`, 164
- `\ddot`, 164
- `\ddots`, 170, 171
- `\DeclareGraphicsExtensions`, 73
- `\DeclareGraphicsRule`, 74
- `\DeclareMathOperator`, 140, 159
- `\DeclareMathOperator*`, 159, 160
- `\DeclareRobustCommand`, 198
- `\def`, 201–205, 210, 211, 220
- `\defaultfontfeatures`, 241
- `\definecolor`, 83
- `\deg`, 158
- `\Delta`, 143
- `\delta`, 143
- `\det`, 158
- `\Diamond`, 178
- `\diamond`, 175
- `\diamondsuit`, 178
- `\digamma`, 142, 143
- `\dim`, 158
- `\displaystyle`, 156
- `\div`, 175
- `\do`, 219
- `\documentclass`, 10, 25, 33, 34, 159, 160, 196, 200, 203, 220, 226, 228, 232
- `\dot`, 164
- `\doteq`, 176
- `\doteqdot`, 176
- `\dotsb`, 154, 165, 171
- `\dotsc`, 171
- `\dotsi`, 171
- `\dotsm`, 171
- `\dotso`, 171
- `\doublerulesep`, 56
- `\Downarrow`, 154, 177
- `\downarrow`, 154, 177
- `\draw`, 76–82, 84–89, 91–93, 95, 96, 98–100, 103–110, 112, 134
- `\edef`, 202
- `\eIf`, 183
- `\ell`, 177, 178
- `\else`, 215
- `\ElseIf`, 182
- `\emph`, 46, 47, 232
- `\emptyset`, 178
- `\encl`, 34
- `\end`, 10–12, 15, 17, 20, 25, 29, 34, 47–51, 53–55, 57, 60–64, 70–72, 76–82, 84–91, 93, 95, 98–100, 103, 104, 107–110, 118–120, 125, 127, 129, 131, 133, 134, 144–149, 151, 157, 162, 163, 165, 170, 172, 181, 183, 184, 187, 196, 197, 200, 203, 206, 214, 215, 218–220, 225–228, 230–232
- `\endcsname`, 202, 203, 206
- `\endfirsthead`, 119, 120
- `\endfoot`, 119, 120
- `\endgroup`, 42
- `\endhead`, 119, 120
- `\endlastfoot`, 119, 120
- `\endlatsfoot`, 119
- `\epsilon`, 41, 143
- `\eqcirc`, 176
- `\equal`, 218
- `\equiv`, 159, 176
- `\eta`, 143
- `\eUlseIf`, 182
- `\euro`, 54
- `\exists`, 178
- `\exp`, 158
- `\expandafter`, 202, 203
- `\extracolsep`, 55
- `\fallingdotseq`, 176
- `\fi`, 215, 220
- `\figure`, 209
- `\fill`, 55, 82, 83, 90, 91, 104
- `\filldraw`, 83, 90
- `\flat`, 178
- `\flushbottom`, 216
- `\footnote`, 46
- `\footnotesize`, 48
- `\For`, 184, 185
- `\ForAll`, 185



`\forall`, 178  
`\ForEach`, 185  
`\foreach`, 104, 105, 108  
`\frac`, 107, 153, 154, 156, 158, 160, 161  
`\framesubtitle`, 226, 227, 232  
`\frametitle`, 226, 227, 232  
`\frenchspacing`, 45  
`\frontmatter`, 15  
`\frown`, 176  
  
`\Gamma`, 143  
`\gamma`, 142, 143, 162  
`\gcd`, 158, 159  
`\geq`, 173, 176, 181  
`\gg`, 176  
`\graphicspath`, 73  
`\grave`, 164  
  
`\H`, 42  
`\hat`, 163, 164, 176  
`\hbar`, 178  
`\hbox`, 43  
`\heartsuit`, 178  
`\hline`, 52, 53, 56  
`\hom`, 158  
`\hookleftarrow`, 177  
`\hookrightarrow`, 177  
`\hphantom`, 50, 174  
`\hspace`, 55  
`\Huge`, 48  
`\huge`, 48  
`\hyphenation`, 57, 58  
  
`\i`, 42  
`\idotsint`, 156, 161  
`\If`, 182, 186  
`\ifnotes`, 215  
`\ifthenelse`, 217, 218  
`\ifx`, 220  
`\iiiint`, 156, 161  
`\iiint`, 156, 161  
`\iint`, 156, 161  
`\Im`, 178  
`\imath`, 176, 178  
`\in`, 174, 176  
`\include`, 17, 29, 213  
`\includegraphics`, 72  
`\includegraphics`, 72–74  
`\includeonly`, 17  
`\index`, 31, 32, 197  
`\inf`, 158  
`\infty`, 147, 155, 156, 178  
`\input`, 17  
`\institute`, 232  
`\int`, 156, 160, 161, 171  
`\intertext`, 149  
`\iota`, 143  
`\isodd`, 218  
`\isundefined`, 218  
`\item`, 59–62, 64, 227, 229–232  
`\itemindent`, 63  
`\itemsep`, 63  
`\itshape`, 49, 188  
  
`\j`, 42  
`\jmath`, 176, 178  
`\Join`, 176  
  
`\kappa`, 143  
`\ker`, 158  
`\kill`, 56, 57  
`\KwData`, 182  
`\KwIn`, 181  
`\KwOut`, 181  
`\KwResult`, 182  
`\KwRet`, 182  
  
`\L`, 43  
`\l`, 43  
`\label`, 17, 18, 20, 70, 118, 144, 145, 147, 167, 204, 205  
`\labelenumi`, 61  
`\labelenumii`, 61  
`\labelenumiii`, 61  
`\labelenumiiv`, 61  
`\labelitemi`, 59, 60  
`\labelitemii`, 60  
`\labelitemiii`, 60  
`\labelitemiv`, 60  
`\labelsep`, 63  
`\labelwidth`, 63  
`\Lambda`, 143  
`\lambda`, 143  
`\langle`, 153, 154, 163  
`\LARGE`, 48

`\Large`, 48  
`\large`, 48  
`\LaTeX`, 10–12, 22, 25, 28, 29, 203  
`\lceil`, 153, 154  
`\ldotp`, 170  
`\ldots`, 153, 170, 171, 173  
`\left`, 149–153, 155, 160–163, 174  
`\Leftarrow`, 177  
`\leftarrow`, 177, 181  
`\leftharpoondown`, 177  
`\leftharpoonup`, 177  
`\leftmargin`, 63, 64  
`\Leftrightarrow`, 177  
`\leftrightharpoonup`, 177  
`\leftroot`, 162  
`\legend`, 71  
`\lElse`, 183  
`\lengthtest`, 218  
`\leq`, 151–153, 157, 176  
`\let`, 204, 205  
`\lfloor`, 153, 154  
`\lg`, 155, 158  
`\lhd`, 175  
`\lim`, 140, 158  
`\liminf`, 158  
`\limsup`, 158  
`\lipsum`, 194  
`\listofalgorithms`, 180  
`\listoffigures`, 29  
`\listoflistings`, 187  
`\listoftables`, 29  
`\listparindent`, 63  
`\ll`, 176  
`\ln`, 158  
`\log`, 155, 158  
`\long`, 202  
`\Longleftarrow`, 177  
`\longleftarrow`, 177  
`\Longleftrightarrow`, 177  
`\longleftrightharpoonup`, 177  
`\longmapsto`, 177  
`\Longrightarrow`, 177  
`\longrightarrow`, 177  
`\lstinputlisting`, 186  
`\lstset`, 188  
`\lVert`, 152, 154  
`\lvert`, 152, 154, 162  
`\mainmatter`, 15  
`\makeatletter`, 203–205  
`\makeatother`, 203–205  
`\makeindex`, 31, 197  
`\MakeRobustCommand`, 198  
`\maketitle`, 10–12, 15, 29, 226, 232  
`\mapsto`, 163, 166, 177  
`\marginpar`, 46  
`\mathbb`, 140, 166, 173, 174  
`\mathbf`, 140, 174  
`\mathcal`, 175  
`\mathfrac`, 140  
`\mathit`, 174  
`\mathop`, 160  
`\mathring`, 164  
`\mathrm`, 118, 120, 159–161, 173, 174  
`\mathsf`, 175  
`\mathtt`, 175  
`\max`, 158  
`\mdseries`, 49  
`\medspace`, 174  
`\metre`, 173  
`\mho`, 178  
`\mid`, 174, 176  
`\midrule`, 53, 54, 118, 120  
`\min`, 158  
`\mod`, 159  
`\mode`, 229  
`\mode*`, 229  
`\models`, 176  
`\MonoIdx`, 197  
`\mp`, 175  
`\mu`, 143  
`\multicolumn`, 53, 54, 120  
`\multimap`, 176  
`\n`, 106  
`\nabla`, 178  
`\natural`, 178  
`\nearrow`, 177  
`\neg`, 178  
`\negmedspace`, 174  
`\negthickspace`, 174

`\negthinspace`, 174  
`\neq`, 176, 181  
`\newbibliography`, 28  
`\newboolean`, 217  
`\newcommand`, 120, 195–197, 200, 206  
`\newcounter`, 63, 64, 213, 214, 219  
`\newenvironment`, 64, 65, 206, 207, 226  
`\newif`, 214, 215  
`\newkeycommand`, 211  
`\newkeyenvironment`, 211  
`\newlength`, 216  
`\newreformat`, 20  
`\newtheorem`, 140, 168, 169  
`\newtheoremstyle`, 169, 170  
`\ni`, 176  
`\nocite`, 23  
`\node`, 96, 109, 110, 133  
`\nodepart`, 96  
`\noexpand`, 202  
`\nonumber`, 147  
`\normalfont`, 49  
`\normalsize`, 48  
`\NOT`, 218  
`\not`, 218, 219  
`\notesttrue`, 215  
`\notin`, 176  
`\nu`, 143  
`\nwarrow`, 177  
  
`\O`, 43  
`\o`, 43, 143  
`\odot`, 175  
`\OE`, 43  
`\oe`, 43  
`\oint`, 156  
`\Omega`, 143  
`\omega`, 143  
`\ominus`, 175  
`\opening`, 34  
`\oplus`, 175  
`\OR`, 218  
`\or`, 218  
`\oslash`, 175  
`\Other`, 184  
  
`\otimes`, 175  
`\overbrace`, 163, 164  
`\overleftarrow`, 164  
`\overleftrightharrow`, 164, 177  
`\overline`, 163, 164  
`\overrightarrow`, 164  
  
`\p`, 106  
`\pageref`, 18  
`\paragraph`, 30  
`\parallel`, 176  
`\parindent`, 216  
`\parsep`, 63  
`\parskip`, 63, 216, 217  
`\part`, 29, 30  
`\part*`, 29  
`\partial`, 161, 178  
`\partopsep`, 63  
`\path`, 77–84, 89–91, 93, 95, 98, 105–107  
`\pause`, 229, 230  
`\per`, 173  
`\perp`, 176  
`\pgfmathparse`, 101  
`\pgfplotsset`, 126  
`\phantom`, 50, 174  
`\Phi`, 143  
`\phi`, 143  
`\Pi`, 143, 232  
`\pi`, 143  
`\pitchfork`, 176  
`\pm`, 175  
`\pmb`, 174  
`\pmod`, 159  
`\pod`, 159  
`\Pr`, 158  
`\prec`, 176  
`\precapprox`, 176  
`\preccurlyeq`, 176  
`\preceq`, 176  
`\precsim`, 176  
`\prettyref`, 20  
`\prime`, 178  
`\printindex`, 31, 197  
`\prod`, 156  
`\propto`, 176  
`\protect`, 70, 198

- \provideboolean, 217
- \ps, 34
- \Psi, 143
- \psi, 143
- \qedhere, 170, 232
- \qqquad, 47, 48, 145–147, 151, 172–174
- \quad, 172, 174
- \raggedleft, 51
- \raggedright, 51
- \rangle, 153, 154, 163
- \ratio, 219
- \rceil, 153, 154
- \Re, 178
- \ref, 17–20, 40, 144, 145, 147, 167
- \refname, 23
- \relay, 205
- \renewcommand, 23, 60, 61, 195, 196
- \renewenvironment, 207
- \Repeat, 185
- \rfloor, 153, 154
- \rhd, 175
- \rho, 143
- \right, 149–153, 155, 160–163, 174
- \Rightarrow, 177
- \rightarrow, 177
- \rightharpoondown, 177
- \rightharpoonup, 177
- \rightleftharpoons, 177
- \rightmargin, 63, 64
- \risingdotseq, 176
- \rmfamily, 49
- \Roman, 61, 62
- \roman, 61, 62
- \rVert, 152, 154
- \rvert, 152, 154, 160, 162
- \scriptsize, 48
- \scshape, 49
- \searrow, 177
- \sec, 158
- \secnumdepth, 30
- \second, 173
- \section, 10, 12, 14, 16, 30, 204, 205, 215, 232
- \section\*, 14
- \selectlanguage, 58
- \setboolean, 217
- \setcounter, 30, 213, 214, 219
- \setkeys, 73, 210, 211
- \setlength, 64, 216, 217, 219
- \setminus, 175
- \setmonofont, 241
- \setromanfont, 241
- \setsansfont, 241
- \settodepth, 217
- \settoheight, 217
- \settowidth, 217
- \sffamily, 49
- \shade, 83
- \shadedraw, 83
- \sharp, 178
- \shortintertext, 149
- \shortmid, 176
- \shortparallel, 176
- \shoveleft, 146
- \shoveright, 146
- \SI, 173
- \Sigma, 143
- \sigma, 143
- \signature, 34
- \sim, 176
- \simeq, 176
- \sin, 140, 158
- \sinh, 158
- \sisetup, 173
- \slshape, 49
- \small, 48
- \smallfrown, 176
- \smallsmile, 176
- \smile, 176
- \spadesuit, 178
- \spy, 108
- \sqcap, 175
- \sqcup, 175
- \sqrt, 161, 162
- \sqsubset, 176
- \sqsubseteq, 176
- \sqsupset, 176
- \sqsupseteq, 176

- \squared, 173
- \ss, 43
- \star, 175
- \stepcounter, 213, 214
- \subparagraph, 30
- \subsection, 30
- \subset, 176
- \subseteq, 176
- \substack, 157
- \subsubsection, 30
- \succ, 176
- \succapprox, 176
- \succcurlyeq, 176
- \succeq, 176
- \succsim, 176
- \sum, 141, 144, 147, 155–157, 173
- \sup, 158
- \supset, 176
- \supseteq, 176
- \surd, 178
- \swarrow, 177
- \Switch, 183, 184
- \t, 42
- \tabcolsep, 55, 56
- \tablename, 120
- \tableofcontents, 3, 15, 29
- \tan, 158
- \tanh, 158
- \tau, 143
- \tcc, 185
- \tcp, 183, 185, 186
- \tcp\*, 186
- \TeX, 22, 27, 28, 31, 196, 203
- \text, 140, 150, 151, 157, 164, 165, 173
- \textasciicircum, 40
- \textasciitilde, 40
- \textbackslash, 39, 40, 197
- \textbf, 41, 42, 49, 54, 62, 118, 120, 127, 129, 131, 133, 134
- \textmdash, 44, 45
- \textendash, 44
- \textheight, 73, 216
- \textit, 49
- \textmd, 49
- \textnormal, 49
- \textrm, 49
- \textsc, 49, 129, 181
- \textsf, 49
- \textsl, 49
- \textstyle, 156
- \texttt, 49, 80, 116, 117, 197
- \textup, 49
- \textvisiblespace, 10
- \textwidth, 73, 134, 216
- \thanks, 11
- \theoremstyle, 168, 169
- \therefore, 176
- \Theta, 143
- \theta, 143
- \thetable, 120
- \thickapprox, 176
- \thicksim, 176
- \thickspace, 174
- \thinspace, 174
- \tikz, 76, 77, 81–83, 85–88, 96, 103, 105, 106
- \tikzset, 77, 102, 103, 107, 108, 111
- \tikztonodes, 107, 108
- \tikztostart, 107, 108
- \tikztotarget, 107, 108
- \tilde, 164
- \times, 76, 150, 151, 165, 172, 175
- \tiny, 48, 49
- \title, 10, 11, 226, 232
- \titlepage, 11
- \to, 155, 158, 160, 166, 171
- \tocdepth, 30
- \top, 178
- \toprule, 53, 54, 118, 120
- \topsep, 63
- \triangle, 178
- \triangleleft, 175
- \triangleright, 175
- \ttfamily, 49, 188
- \u, 42
- \uCase, 184
- \uElseIf, 182, 183

$\backslash$ uIf, 182, 183  
 $\backslash$ underbar, 164  
 $\backslash$ underbrace, 163–165  
 $\backslash$ underleftarrow, 164, 177  
 $\backslash$ underleftrightarrow, 164, 177  
 $\backslash$ underline, 164  
 $\backslash$ underrightarrow, 164, 177  
 $\backslash$ unlhd, 175  
 $\backslash$ unrhd, 175  
 $\backslash$ Uparrow, 154, 177  
 $\backslash$ uparrow, 154, 177  
 $\backslash$ Updownarrow, 154, 177  
 $\backslash$ updownarrow, 154, 177  
 $\backslash$ uplus, 175  
 $\backslash$ uproot, 162  
 $\backslash$ upshape, 49  
 $\backslash$ Upsilon, 143  
 $\backslash$ upsilon, 143  
 $\backslash$ usecounter, 64  
 $\backslash$ usepackage, 10, 11, 20, 22, 25, 34, 58, 62, 140, 169, 197, 210, 218, 219, 228, 241  
 $\backslash$ usetHEME, 232  
 $\backslash$ usetikzlibrary, 89, 100, 110, 135  
 $\backslash$ v, 42  
 $\backslash$ value, 218  
 $\backslash$ varDelta, 143  
 $\backslash$ varepsilon, 142, 143  
 $\backslash$ varGamma, 143  
 $\backslash$ varkappa, 143  
 $\backslash$ varLambda, 143  
 $\backslash$ varOmega, 143  
 $\backslash$ varPhi, 143  
 $\backslash$ varphi, 143  
 $\backslash$ varPi, 143  
 $\backslash$ varpi, 143  
 $\backslash$ varpropto, 176  
 $\backslash$ varPsi, 143  
 $\backslash$ varrho, 142, 143  
 $\backslash$ varSigma, 143  
 $\backslash$ varsigma, 143  
 $\backslash$ varTheta, 143  
 $\backslash$ vartheta, 142, 143  
 $\backslash$ varUpsilon, 143  
 $\backslash$ varXi, 143  
 $\backslash$ Vdash, 176  
 $\backslash$ vDash, 176  
 $\backslash$ vdash, 176  
 $\backslash$ vdots, 170, 171  
 $\backslash$ vec, 164  
 $\backslash$ vee, 175  
 $\backslash$ vert, 152  
 $\backslash$ vline, 52, 56  
 $\backslash$ vphantom, 50, 151  
 $\backslash$ Vvdash, 176  
 $\backslash$ wedge, 175  
 $\backslash$ While, 181, 185  
 $\backslash$ whiledo, 218, 219  
 $\backslash$ widehat, 164  
 $\backslash$ widetilde, 163, 164  
 $\backslash$ widthof, 219  
 $\backslash$ wp, 178  
 $\backslash$ wr, 175  
 $\backslash$ x, 106  
 $\backslash$ xhookleftarrow, 177, 178  
 $\backslash$ xhookrightarrow, 177, 178  
 $\backslash$ Xi, 143  
 $\backslash$ xi, 143  
 $\backslash$ xLeftarrow, 177, 178  
 $\backslash$ xleftarrow, 177  
 $\backslash$ xleftharpoondown, 177, 178  
 $\backslash$ xleftharpoonup, 177, 178  
 $\backslash$ xLeftrightarrow, 177, 178  
 $\backslash$ xleftrightharpoonup, 177, 178  
 $\backslash$ xleftrightharpoons, 177, 178  
 $\backslash$ xmapsto, 177, 178  
 $\backslash$ xrightarrow, 177, 178  
 $\backslash$ xrightarrow, 177  
 $\backslash$ xrightharpoondown, 177, 178  
 $\backslash$ xrightharpoonup, 177, 178  
 $\backslash$ xrightleftharpoons, 177, 178  
 $\backslash$ xticklabels, 128  
 $\backslash$ y, 106  
 $\backslash$ yticklabels, 128  
 $\backslash$ zeta, 143

## Index of Environments

- abstract, 12
- algorithm, 179, 180
- algorithm\*, 180
- algorithm2e, 179, 181, 183, 184
- align, 144, 147, 148
- align\*, 144, 149, 151, 172
- aligned, 149
- alignedat, 149
- array, 51, 54, 56, 162
- axis, 125–129, 131, 133–135
  
- Bmatrix, 162
- bmatrix, 162
  
- cases, 151, 165
- center, 50
  
- description, 62
- document, 10–12, 15, 17, 20, 25, 29, 34, 159, 160, 196, 197, 200, 203, 206, 214, 215, 218–220, 226, 228, 232
  
- enumerate, 61, 62, 232
- eqnarray, 150
- equation, 144–146
- equation\*, 144, 145, 149
  
- figure, 69–72, 180
- figure\*, 70, 72
- flushleft, 51
- flushright, 51
- footnotesize, 48
- frame, 194, 225–232
- fullpage, 71
- function, 180, 181
- function\*, 181
  
- gather, 145, 147
- gathered, 149
  
- Huge, 48, 49
- huge, 48
  
- itemize, 59–61, 227, 230, 231
  
- LARGE, 48
- Large, 48
- large, 48
- leftfullpage, 71
- letter, 34
- list, 63–65
- longtable, 119, 120
- lstlisting, 186, 187
- lstlistings, 187
  
- matrix, 162
- multline, 146
  
- normalsize, 48, 49
  
- pgfplots, 123
- pmatrix, 162
- procedure, 180
- procedure\*, 180
- proof, 140, 166, 170
  
- quotation, 47
- quote, 47
  
- scope, 103, 104
- scriptsize, 48
- sidewaysfigure, 119
- sidewaystable, 119
- small, 48
- smallmatrix, 163
- split, 145–147
- subarray, 157
- substack, 157
  
- tabbing, 50, 56, 57, 162, 179
- table, 113, 118, 180
- table\*, 118
- tabular, 50–54, 56, 118–120
- tabular\*, 51, 54–56
- theorem, 232
- tikzpicture, 75–82, 84, 85, 89–93, 95, 98–100, 103, 104, 107–110, 125, 127, 129, 131, 133, 134
- tiny, 48
- titlepage, 11

tt, 57

verse, 47, 48

Vmatrix, 162

vmatrix, 162



## Index of Classes

acmcls, 140  
amscs, 140  
article, 3, 10, 11, 23, 25, 30,  
33, 194, 196, 200, 220  
  
beamer, 194, 225, 226, 228–230,  
271  
book, 30, 33  
  
letter, 14, 33, 34  
  
memoir, 35  
  
report, 33

## Index of Packages

- amsmath**, 154
- algorithm2e, 179–185
- amsbsy, 140
- amscd, 140
- amsfonts, 140
- amsmath, 35, 139, 140, 143–146, 149, 150, 154, 156, 158–162, 169, 171, 175
- amsopn, 140
- amssymb, 140
- amstext, 140
- amsthm, 140, 166, 169, 170
- arrows, 88, 89
- babel, 58
- beamer, 75
- beamerarticle, 227
- bibtopic, 28
- bibunits, 29
- bidl, 58
- booktabs, 53, 54, 118
- calc, 100, 219
- calctab, 121, 194
- ccaption, 71
- chapterbib, 29
- circuits.logic.CDH, 110
- circuits.logic.IEC, 110
- circuits.logic.US, 110
- classicthesis, 35
- colortbl, 53
- cool, xi, 158, 160
- coverpage, 35
- csvpie, 125
- datatool, 120
- dpfloat, 71
- enumerate, 59, 61, 62
- esint, 160, 161
- etoolbox, 243
- fancyhdr, 35
- fontspec, 4, 237, 240, 241
- fourier, 35
- graphics, 74
- graphicx, 35, 72
- ifthen, 213, 217–219
- keycommand, 209, 211
- keyval, 35, 72, 209–211
- lastpage, 35
- lipsum, 194
- listings, 35, 186, 188
- longtable, 119
- makerobust, 198
- mathdesign, 175
- mathptmx, 10, 11
- mathtools, 35, 149
- multibbl, 28
- multibib, 28
- multind, 30, 31
- named, 22, 25
- natbib, vii, 26, 27
- pgf, 69, 75
- pgfkeys, 102
- pgfplot, 135
- pgfplots, 69, 125, 126, 128, 130, 131, 135
- pgfplotstable, 120
- polyglossia, 58
- prettyref, 19, 20, 35, 145
- rotating, 119
- siunitx, 173
- spreadtab, 121
- tikz, 69, 75, 76, 78, 83, 95, 97, 99, 100, 102, 109, 225, 230, 244, 271
- tkz-berge, 245
- url, 35
- xcolor, 83, 84
- xkeyval, 35

## Index of Commands and Languages

- `./install-tl.sh`, 237, 238
- ANSI C, 186
- ANSI C++, 186
- `apt-get`, 237, 241
- `bibtex`, 3, 24–26, 29, 31
- `cd`, 238
- Debian, 241
- `debian`, 4
- `dvipdf`, 7
- `dvips`, 7
- `eclipse`, 4, 6
- Eiffel, 186
- `emacs`, 6, 8
- `epstopdf`, 74
- `excel`, 124
- `export`, 238, 239
- `fc-cache`, 240
- GhostScript, 74
- `ghostview`, 245
- `gimp`, 74
- `gnuplot`, 71, 74
- `gs`, 74
- `gsview`, 245
- HTML, 186
- `install-tl.sh`, 237
- `ispell`, 58
- Java, 186
- `latex`, 6–8, 12, 13, 16–19, 24, 25
- Linux, 4
- `lisp`, xvii, 220
- `luatex`, 240
- Lyx, 245
- `makeindex`, 31
- `matlab`, 125
- `metapost`, 271
- MikTeX, 245
- `mkdir`, 237
- `mount`, 237
- `mpost`, 74
- `pdflatex`, 3, 7, 8, 108
- PHP, 186
- Python, 186
- `rmdir`, 238
- `texdoc`, 27, 34
- `texhash`, 239
- Texniccenter, 245
- Ubuntu, 241
- `ubuntu`, 4
- `umount`, 238
- Unix, 4
- `vim`, 6–8, 58
- `xdvi`, 7
- `xelatex`, 4, 43, 45, 58, 108
- `xetex`, 240
- XML, 186



# Acronyms

<b>AMS</b>	American Mathematical Society
<b>APL</b>	A Programming Language
<b>CTAN</b>	Comprehensive T <sub>E</sub> X Archive Network
<b>CD</b>	Compact Disk
<b>FAQ</b>	Frequently Asked Questions
<b>GUI</b>	Graphical User Interfaces
<b>IDE</b>	Integrated Development Environment
<b>ISBN</b>	International Standard Book Number
<b>SI</b>	International System of Units
<b>TUG</b>	T <sub>E</sub> X Users Group
<b>WYSIWYG</b>	What You See is What You Get



# Bibliography

- Abrahams, P. A., K. A. Hargreaves and K. Berry [2003]. *TEX for the Impatient*. This book is freely available from <ftp://tug.org/tex/impatient>. Addison–Wesley.
- Adriaens, Henri [2008]. *The xkeyval Package*. Version v2.6a.
- Allan, Robert [2001]. *Punctuation*. Oxford University Press. ISBN: 0-19-860439-4.
- American Mathematical Society [2002]. *User's Guide for the amsmath Package*. Version 2.0.
- [2004]. *Using the amsthm Package*. Version 2.20.
- Arseneau, Donald [2010]. *url.sty Version 3.3*. Edited as a L<sup>A</sup>T<sub>E</sub>X document by Robin Fairbairns.
- Aslaksen, Helmer [1993]. “Ten T<sub>E</sub>X Tricks for the Mathematician”. In: *TUGboat, Communications of the T<sub>E</sub>X Users Group* 14. A modern version of this paper is available from <http://www.math.nus.edu.sg/aslaksen/cs/tug-update.pdf>, pp. 135–136.
- Bigwood, Sally and Melissa Spore [2003]. *Presenting Numbers, Tables, and Charts*. Oxford University Press. ISBN: 0-19-860722-9.
- Bovani, Michel [2005]. *Fourier GUTenberg*.
- Braams, Johannes et al. [2001]. *The L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> Sources*. URL: <http://www.tug.org/texlive/Contents/live/texmf-dist/doc/latex/base/source2e.pdf>.
- Breitenbucher, Jon [2005]. “L<sup>A</sup>T<sub>E</sub>X at a Liberal Arts College”. In: *The PracT<sub>E</sub>X Journal* 3. URL: <http://www.tug.org/pracjournal/index.html>.
- Buchsbaum, Arthur and Francisco Reinaldo [2007]. “A Tool for Logicians”. In: *The PracT<sub>E</sub>X Journal* 3. URL: <http://www.tug.org/pracjournal/index.html>.
- Burt, John [2005]. “Using poemscol for Critical Editions of Poetry”. In: *The PracT<sub>E</sub>X Journal* 3. URL: <http://www.tug.org/pracjournal/index.html>.
- Carlisle, D. P. [2003]. *Packages in the graphics Bundle*.
- Carlisle, D. P. and S. P. Q. Ratz [1999]. *The graphicx Package*.
- Carlisle, David [1999a]. *The enumerate Package*.
- [1999b]. *The keyval Package*.
- Chervet, Florent [2009]. *The keycommand Package*.
- Dearborn, Elizabeth [2006]. “T<sub>E</sub>X for Medicine”. In: *The PracT<sub>E</sub>X Journal* 4. URL: <http://www.tug.org/pracjournal/index.html>.

- Eijkhout, V. [2004]. *The Computer Science of T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X*. URL: <http://www.cs.utk.edu/~eijkhout/594-LaTeX/handouts/TeX%20LaTeX%20course.pdf>.
- [2007]. *T<sub>E</sub>X by Topic, A T<sub>E</sub>Xnician's Reference*. This book is freely available from <http://www.eijkhout.net/tbt/>. Addison-Wesley. ISBN: 0-201-56882-9.
- Fenn, Jürgen [2006]. “Managing Citations and Your Bibliography with BIB<sub>T</sub>E<sub>X</sub>”. In: *PracT<sub>E</sub>X Journal*. URL: <http://www.tug.org/pracjourn/2006-4/fenn/>.
- Feuersänger, Christian [2008]. *Manual for Package PGFPLOTSTABLE*. Version 1.1.
- [2010]. *Manual for Package PGFPLOTS*. Version 1.3.
- Fine, Jonathan [1992]. “Some Basic Control Macros for T<sub>E</sub>X”. In: *TUGboat* 13.1.
- Firio, Christophe [2004]. *algorithm2e.sty — package for algorithms* release 3.3.
- Garcia, Aracele and Arthur Buchsbaum [2010]. “About L<sup>A</sup>T<sub>E</sub>X tools that students of Logic should know”. In: *The PracT<sub>E</sub>X Journal* 1. In Portuguese. URL: <http://www.tug.org/pracjourn/index.html>.
- Giacomelli, Roberto [2009]. *calctab Package*.
- Goldberg, Jeffrey P. [2010]. *The lastpage Package*.
- Goossens, M., F. Mittelbach and A. Samarin [1994]. *The L<sup>A</sup>T<sub>E</sub>X Companion*. Addison-Wesley. ISBN: 0-201-54199-8.
- Goossens, M., S. Rahtz and F. Mittelbach [1997]. *The L<sup>A</sup>T<sub>E</sub>X Graphics Companion: Illustrating documents with T<sub>E</sub>X and PostScript*. Addison-Wesley. ISBN: 0-201-85469-4.
- [1999]. *The L<sup>A</sup>T<sub>E</sub>X Web Companion: Integrating T<sub>E</sub>X, HTML, and XML*. Addison-Wesley. ISBN: 0-201-43311-7.
- Graham, R.L., D.E. Knuth and O. Patashnik [1989]. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley. ISBN: 0-201-14236-8.
- Hagen, Hans. *Making MetaPost Outlines*.
- [2002]. *META<sub>F</sub>UN*. URL: <http://wiki.contextgarden.net/MetaFun>.
- Heck, André [2005a]. *Learning L<sup>A</sup>T<sub>E</sub>X by Doing*. URL: <http://www.science.uva.nl/onderwijs/lesmateriaal/latex/latexcourse.pdf>.
- [2005b]. *Learning META<sub>P</sub>OST by Doing*. URL: <http://remote.science.uva.nl/~heck/Courses/mptut.pdf>.
- Heinz, Carsten and Brooks Moses [2007]. *The Listings Package*. Version 1.4.
- Høgholm, Marten [2010]. *The mattools Package*.
- Hurlin, Clément [2007]. *Practical Introduction to META<sub>P</sub>OST*. URL: <http://www-sop.inria.fr/everest/personnel/Clement.Hurlin/misc/Practical-introduction-to-MetaPost.pdf>.
- Kern, Uwe [2007]. *Extending L<sup>A</sup>T<sub>E</sub>X's color facilities: xcolor*.



- Knuth, D. E. [1990]. *The T<sub>E</sub>Xbook*. The source of this book is freely available from <http://www.ctan.org/tex-archive/systems/knuth/tex/>. Addison–Wesley. ISBN: 0-201-13447-0.
- Krab Thorub, Kresten, Frank Jensen and Chris Rowley [2005]. *The calc Package* Infix notation arithmetic in L<sup>A</sup>T<sub>E</sub>X.
- Krishnan, E., ed. [2003]. *L<sup>A</sup>T<sub>E</sub>X Tutorials A Primer*. URL: <http://www.tug.org.in/tutorials.html>.
- Lamport, L. [1994]. *L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System*. Addison–Wesley. ISBN: 0-021-52983-1.
- Maltby, Gavin [1992]. *An Introduction to T<sub>E</sub>X and Friends*. URL: <http://citeseer.ist.psu.edu/maltby92introduction.html>.
- Miede, André [2010]. *The Classic Thesis Style*.
- Mühlich, Matthias [2006]. *The CoverPage Package*. Version 1.01.
- Oetiker, Tobias et al. [2007]. *The Not so Short Introduction to L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>*. URL: <http://tobi.oetiker.ch/lshort/>.
- Oostrum, Piet van [2004]. *Page Layout in L<sup>A</sup>T<sub>E</sub>X*.
- Pakin, Scot [2005]. *The Comprehensive L<sup>A</sup>T<sub>E</sub>X Symbol List*. URL: <http://tug.ctan.org/info/symbols/comprehensive/symbols-letter.pdf>.
- [2006]. *The Visual L<sup>A</sup>T<sub>E</sub>X FAQ*. URL: <http://http://tug.ctan.org/tex-archive/info/visualFAQ>.
- Peyton Jones, S. and J. Hughes [1999]. Haskell 98: *A Non-strict, Purely Functional Language*. <http://www.haskell.org/onlinereport/>.
- Peyton Jones, Simon L., John Huges and John Launchberry [1993]. “How to Give a Good Research Talk”. In: *ACM SIGPLAN Notices* 28.11, pp. 9–12. URL: <http://research.microsoft.com/en-us/um/people/simonpj/papers/giving-a-talk/giving-a-talk.htm>.
- Reckdahl, Keith [2006]. *Using Imported Graphics in L<sup>A</sup>T<sub>E</sub>X and pdfL<sup>A</sup>T<sub>E</sub>X*. URL: <ftp://ftp.tex.ac.uk/tex-archive/info/epslatex.pdf>.
- Rein, Hanno. *L<sup>A</sup>T<sub>E</sub>X Coffee Stains*. URL: <http://hanno-rein.de/archives/349>.
- Robertson, Will [2008]. *The fontspec Package*.
- Ruland, Kevin S. [2007]. *Improved Reference Formatting for L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>*.
- Schattschneider, Doris [1990]. *M. C. Escher Visions of Symmetry*. New York: W. H. Freeman and Company. ISBN: 0-7167-2126-0.
- Sedgewick, R. and P. Flajolet [1996]. *An Introduction to the Analysis of Algorithms*. Addison–Wesley Publishing Company. ISBN: 0-201-40009-X.
- Senthil, Kumar M. [2007]. “L<sup>A</sup>T<sub>E</sub>X Tools for Life Scientists (BioT<sub>E</sub>Xniques?)” In: *The PracT<sub>E</sub>X Journal* 4. URL: <http://www.tug.org/pracjourn/index.html>.
- Talbot, Nicola [2007]. *datatool v 1.01: Database and data manipulation*.
- Tantau, Till [2010]. *TikZ & PGF*. Manual for Version 2.00-cvs.
- Tantau, Till, Joseph Wright and Vedran Miletic [2010]. *The BEAMER Class*. Version 3.10.
- Tellechea, Christian [2010]. *spreadtab v0.3*.

- The  $\text{\LaTeX}$ 3 Project [2001a].  *$\text{\LaTeX}$ 2 $\epsilon$  Font Selection*. URL: <http://www.latex-project.org/guides/fntguide.tex>.
- [2001b].  *$\text{\LaTeX}$ 2 $\epsilon$  for Authors*. URL: <http://www.latex-project.org/guides/usrguide.tex>.
- [2001c].  *$\text{\LaTeX}$ 2 $\epsilon$  for Class and Package Writers*. URL: <http://www.latex-project.org/guides/clsguide.tex>.
- The  $\text{\LaTeX}$ 3 Project [2001]. *Configuration Options for  $\text{\LaTeX}$ 2 $\epsilon$* . URL: <http://www.latex-project.org/guides/cgfguide.tex>.
- Thomson, A. J. and A. V. Martinet [1986a]. *A Practical English Grammar*. Fourth Edition. Oxford University Press. ISBN: 0-19-431342-5.
- [1986b]. *A Practical English Grammar*, Exercises 1. Third Edition. Oxford University Press. ISBN: 978-0-19-431343-8.
- [1986c]. *A Practical English Grammar*, Exercises 2. Third Edition. Oxford University Press. ISBN: 978-0-19-431344-5.
- Thomson, Paul A. [2008a]. “Clinical Trials Management on the Internet — I. Using  $\text{\LaTeX}$  and SAS to Produce Customized Forms”. In: *The Prac $\text{\TeX}$  Journal* 3.
- [2008b]. “Clinical Trials Management on the Internet — II. Using  $\text{\LaTeX}$ , PostScript, and SAS to Produce Barcode Label sheets”. In: *The Prac $\text{\TeX}$  Journal* 3. URL: <http://www.tug.org/pracjourn/index.html>.
- Trask, R. L. [1997]. *Penguin Guide to Punctuation*. Penguin Books. ISBN: 0-140-51366-3.
- Veytsman, Boris and Leila Akhmadeeva [2006]. “Drawing Medical Pedigree Trees with  $\text{\TeX}$  and PSTricks”. In: *The Prac $\text{\TeX}$  Journal* 4. URL: <http://www.tug.org/pracjourn/index.html>.
- Voß, Herbert [2008]. *Tabellen mit  $\text{\LaTeX}$* . Lehmanns / Dante e.V. ISBN: 9783865412591.
- [2009]. *Math Mode — v. 2.43*. CHECK THIS.
- Wilson, Peter [2010]. *The Memoir Class*. Maintained by Lars Madsen.
- Wright, Joseph [2008]. *siunitx — A Comprehensive (SI) units package*.

# Acknowledgements

THIS BOOK would not have been possible without the help of many. First of all, I should like to express my gratitude to Don Knuth for writing  $\text{\TeX}$  and to Leslie Lamport for writing  $\text{\LaTeX}$  — without them the landscape of typesetting would have been dominated by Bill. I should like to Hans Hagen for much `metapost` inspiration. I am extremely grateful to Till Tantau for writing his beautiful `tikz` package and his `beamer` class. Both of them are stars in terms of functionality, productivity, and documentation. I should like to acknowledge David Farley for letting me include the cartoon which is depicted in Figure 4.2. More cartoons like this may be found on <http://ibiblio.org/Dave/>. I should like to thank Rik Kabel, Luca Merciadri, Oleg Paraschenk, and Joseph Wright for useful comments on an early draft. Their comments have been more than helpful. Finally, I should like to thank all those who have worked on  $\text{\LaTeX}$  and friends, all those who have supported  $\text{\LaTeX}$  and friends, and all who have answered all my  $\text{\LaTeX}$  and `METAPOST` questions over the last two decades or so. The following are but a few: André Heck, Barbara Beeton, Cristian Feuersänger, Dan Luecking, David Carlisle, David Kastrup, Denis Roegel, Donald Arseneau, D.P. Story, Frank Mittelbach, Hans Hagen, Heiko Oberdiek, Jim Hefferon, John Hobby, Jonathan Fine, Jonathan Kew, Kees van der Laan, Keith Reckdahl, Kjell Magne Fauske, Mark Wibrow, Nelson Beebe, Peter Flynn, Peter Wilson, Philipp Lehman, Rainer Schöpf, Robin Fairbairns, Ross Moore, Scot Pakin, Stephan Hugel, Taco Hoekwater, Thomas Esser, Ulrike Fisher, Victor Eijkhout, Vincent Zoonekynd, Will Robertson, and all the many, many others. Without them the  $\text{\TeX}$  community would have been much worse off.



## Colophon

THIS BOOK was typeset with pdf<sup>Ⓟ</sup>TeX. The main text was typeset using the book class using *Adobe Garamond Premier Pro* at 12pt as the main font of the text and *Anonymous Pro* for the monospaced font. The page, figure, and table layout was achieved with a user-defined class. The mathdesign, ams-math, and amssymb packages were used to help typeset the mathematics. The bibliography was typeset with the biblatex package. The coverpage was drawn with the help of a little tikzfigure that includes a couple of METAPOST-generated pictures. The pictures are based on a drawing by Escher [Schattschneider, 1990, Drawing 109<sup>II</sup> on Page 207]. The source code of a similar METAPOST program may be found at <http://cswb.ucc.ie/~dongen/mpost/Escher109.html>. The microtype package was used with the options expansion=true and protrusion=true. You may download the most recent version of this book from <http://cswb.ucc.ie/~dongen/LaTeX-and-Friends.pdf>.