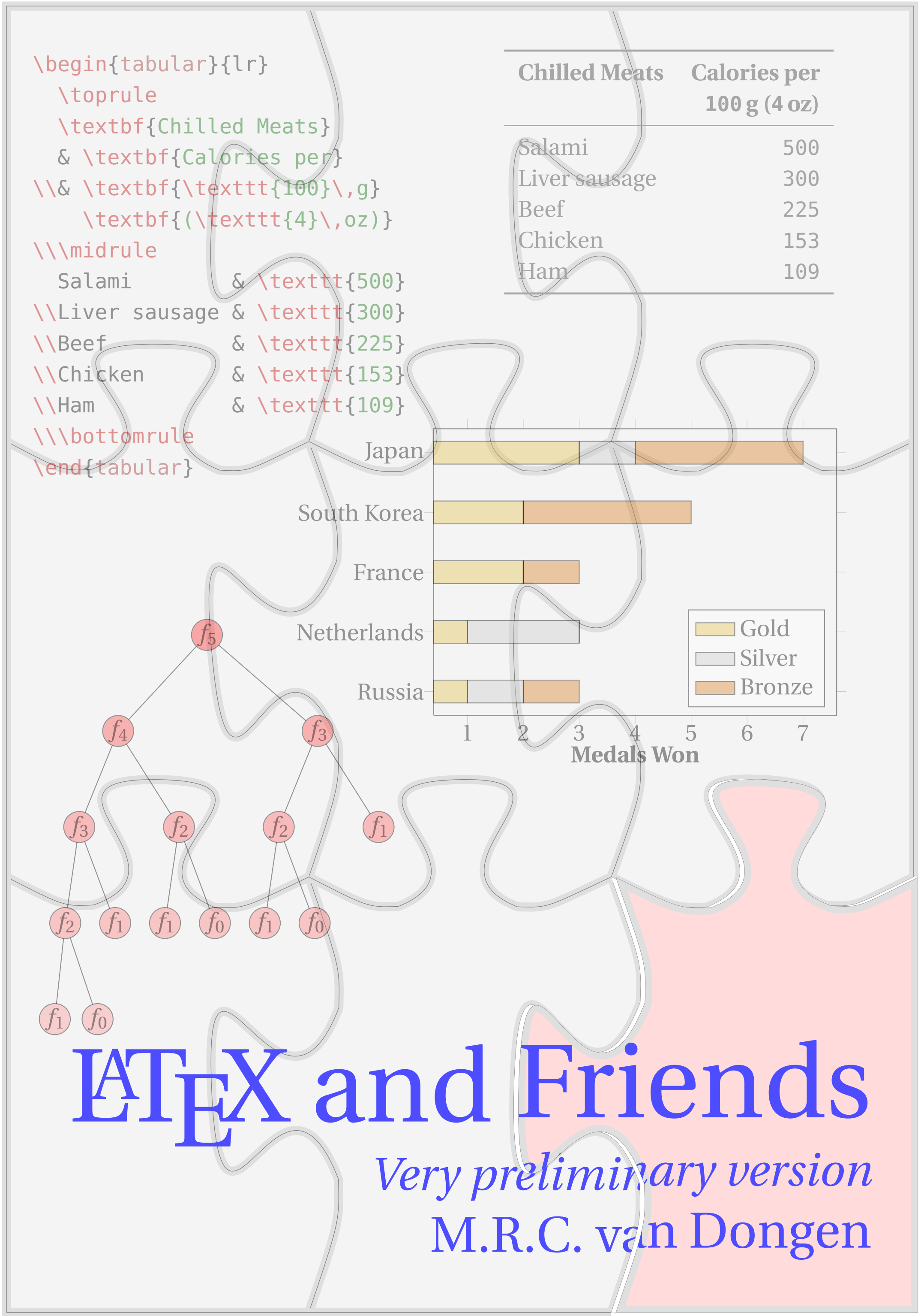
# LaTeX and Friends

*Very preliminary version*

M.R.C. van Dongen



# Preface

This half-finished book is still in preparation. The finished version will have two main parts. The first main part provides computer science graduate (or equivalent) students an introduction to technical writing and presenting with 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. The second main part of this book provides an introduction to academic publishing, writing, and effective presentation of results. This part is still pending.

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 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 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

# List of Figures

# List of Tables

# Part I

# Basics

CHAPTER

1

# Introduction to LaTeX

This chapter is an introduction to LaTeX and friends but it is *not* about typesetting fancy things. Typesetting fancy things is dealt with in further chapters. The main purpose of this chapter is to provide an understanding of the *basic* mechanisms of LaTeX, using plain text as a vehicle. After reading this chapter you should know how to:

- Write a simple LaTeX 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.** *LaTeX 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. LaTeX's markup* *helps you concretise the purpose of your writing and present it in a consistent manner. As a matter of fact, LATEX* 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 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 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 LATEX on different operating systems.

## 1.1 Pros and Cons

Before we start, it is good to look at arguments in favour of 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 LATEX.

- LATEX is difficult. It may take one to several months to learn. True, learning LATEX does take a while. However, it will save you time in the long run, even if you're writing a minor thesis.
- LATEX is not a What You See is What You Get (WYSIWYG) wordprocessor. Correct, but there are many LATEX Integrated Development Environments (IDEs) and some IDEs such as `eclipse` have 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 any need 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 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 LATEX packages, which makes it difficult to find the right package. Agreed, but most LATEX 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`.
- LATEX 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 LATEX [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 created their own websites: `http://theotex.blogspot.com/`, `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 LaTeX:

- LaTeX 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 LaTeX. In addition they provide style and class files which guarantee documents conforming to the required formatting guidelines.
- LaTeX is a Turing-complete programming language. This gives you almost complete control.
- With LaTeX 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.
- LaTeX 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 LaTeX to `html/ps/pdf/DocBook` ....
- LaTeX automatically numbers your chapters, sections, figures, and so on. In addition it provides cross-referencing support.
- LaTeX 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/`), TeXmacs (`http://www.texmacs.org/`), .... Furthermore, some editors and IDEs provide support for LaTeX, e.g. `vim`, `emacs`, `eclipse`, ....
- LaTeX is *very* stable, free, and available on many platforms.
- There is a very large, active, and helpful TeX/LaTeX user-base. Good starting points are listed in Section 20.7.
- LaTeX has comments.
- Most importantly: LaTeX is fun!

## 1.2 Basics

LaTeX [Lamport, 1994] was written by Leslie Lamport as an extension of Donald Knuth's 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 *and* the content of your documents. The following three steps explain how you use LaTeX.

1. You write your document in a 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 TeX Processors

Roughly speaking LaTeX is built on top of TeX. This adds extra functionality to TeX and makes writing your document much easier. LaTeX being built on top of TeX, the result is a TeX program. You get a good understanding of LaTeX by studying TeX's four *processors*, which are basically run in a pipeline [Knuth, 1990; Eijkhout, 2007; Abrahams *et al.*, 2003]. The following are the main functions of 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 LaTeX works, it's time to study the programs you need to turn your input files into readable output. This section may be ignored 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 LaTeX 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 LaTeX syntax, let's see how you turn an existing LaTeX source file into a `.dvi` file. To this end, let's assume you have an error-free LaTeX source file which is called '⟨document⟩`.tex`'. The following command turns your source file into an output file called '⟨document⟩`.dvi`'.

Unix Session
```
$ latex ⟨document⟩.tex
```

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

Unix Session
```
$ latex ⟨document⟩
```

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

```
$ xdvi ⟨document⟩.dvi &                                    Unix Session
```

Now that you have the `.dvi` version of your LaTeX program, you may convert it to other formats. The following converts `⟨document⟩.dvi` to postscript (`⟨document⟩.ps`).

```
$ dvips -o ⟨document⟩.ps ⟨document⟩.dvi                    Unix Session
```

The following converts `⟨document⟩.dvi` to portable document format (`.pdf`).

```
$ dvipdf ⟨document⟩.dvi                                    Unix Session
```

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

```
$ pdflatex ⟨document⟩.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 I use to edit the LaTeX 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.*[1] *This lets you run `latex` from within your editor on the file you're editing. Most `Linux` Graphical User Interfacess (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, LaTeX may be viewed as a program or a language. When referring to the language this book usually uses 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 LaTeX program it usually means LaTeX source file.

### 1.2.4 Staying in Sync

LaTeX sometimes needs more than a single run before it can produce 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 LaTeX may be out of sync. You should rerun LaTeX when this happens. LaTeX outputs a warning when it suspects this is required:

[1] The `emacs` program should let you to do similar things.

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

### 1.2.5 Writing a LaTeX Input Document

LaTeX is a markup language and document preparation system. It forces you to focus on the content and *not* on the presentation. In a 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.

**Content:** The content of your document is determined in terms of text and logical markup. LaTeX 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 LaTeX 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, LaTeX 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 LaTeX 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 quicker. 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/redo" this makes locating the cause of errors very easy.

**Figure 1.1**: Typical 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}
␣␣␣␣␣The␣start.
␣␣\section{Conclusion}
␣␣␣␣␣The␣end.
\end{document}
```

Figure 1.1 depicts a typical example of a LaTeX input program. For this example all spaces in the input have been made explicit by typeseting them with the symbol '␣', which represents a single space. The symbol '␣' 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 till 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 LaTeX that your document should be typeset using the rules determined by the `article` document class.

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

You can only have one document class per 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 the 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 LaTeX input file.

In our example, the `\documentclass` command also takes *optional* arguments. Optional arguments are passed inside square brackets. The optional arguments go before the required argument (this is standard). Optional arguments are called *optional* because they may be omitted. If you omit them then you also 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 packages called mathptmx.

```
\usepackage{mathptmx}
```
LATEX Usage

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}
```
LATEX Usage

The command \LaTeX in the argument of the \title command is for typesetting LATEX. The purpose of the *tilde* (~) is explained further on in this chapter.

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}
```
LATEX Usage

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

```
\author{Sinead\thanks{You're a luvely audience.}}
```
LATEX Usage

You can also build your own titlepage with the titlepage environment. This command can be used only after the \begin{document}. Using the titlepage environment gives you complete control *and* responsibility.

```
\begin{document}
  \begin{titlepage}
      ...
  \end{titlepage}
  ⋮
\end{document}
```
LATEX Usage

For the article class, as well as for most other LATEX 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.

Usually — and this is good practice — all definitions and configurations are put 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 this 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.

LaTeX Usage

```
\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 can specify the abstract as follows.

LaTeX Usage

```
\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.[2] 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:

- A 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.

[2]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 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 second sentence of this paragraph ends in the word ‘elephant.’
This is the first sentence of the second paragraph. The second sentence of this paragraph ends in the word ‘ant’.

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.

## 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 improve 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.*

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 paragraph. Here paragraph is a small unit in a section.

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

None of these sectional units are available in the `letter` class. For each sectional unit LaTeX provides a command 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.

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

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.

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

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.

LaTeX Usage
```
\appendix
\chapter{Proof of Main Theorem}
   \section{A Useful Lemma}
```

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.

LATEX Usage
```
\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. It is explained in Section 1.6. The command `\addcontentsline` inserts an entry for the bibliography in the table of contents.

Notice that the layout of the LATEX 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 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 LATEX 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.*

## 1.4 Document Management

Your 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:[3]

**IDE:** Use a dedicated 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.

[3] If you know other solutions then I'd like to learn from you.

**Figure 1.3**: Using `\include` and `\includeonly`.

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

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

**Files:** LaTeX has commands which let you include input from other files. By putting the contents of each top-level sectional unit in your LaTeX document in a separate file, you can also make the structure in your document more explicit, making it much easier to see the structure.

LaTeX 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 LaTeX 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, LaTeX 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. This is studied in Section 1.6.

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

**Figure 1.4**: The `\label` and `\ref` commands.

```
\chapter{Introduction}
  A short conclusion is
   presented in
   Chapter~\ref{TheEnd}.
\chapter{Conclusion}
\label{TheEnd}
```

**1 Introduction**

A short conclusion is presented in Chapter 2.

**2 Conclusion**

**Figure 1.5**: The `\pageref` command.

```
\chapter{Introduction}
  A short conclusion is
   presented in
   Chapter~\ref{TheEnd}.
  The conclusion starts on
   Page~\pageref{TheEnd}.
\chapter{Conclusion}
\label{TheEnd}
```

**1 Introduction**

A short conclusion is presented in Chapter 2. The conclusion starts on Page 1.

**2 Conclusion**

**`\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` environent it is that `theorem` environment, inside a given `figure` environent 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 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. Tieing 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 referenced 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 LaTeX he decided that a two-pass system was too time consuming. Instead he decided to compromise:

- Label defininitions 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 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 has made an error. However, running LaTeX once more should usually solve the problem as this gets rid of the warning message as well as the question marks.

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 of sections, subsections, and subsubsections, and so on.
2. You choose a unique prefix for the labels of each class. 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.
3. You use the `\newrefformat` 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, `\newrefformat{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, `\newrefformat{id}{\ref{#1}}` gives you the same reference you get by applying `\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.

**Figure 1.6**: Using the `prettyref` package.

```
\usepackage{prettyref}
\newrefformat{ch}{Chapter~\ref{#1}}
\newrefformat{sec}{Section~\ref{#1}}
\newrefformat{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}
```

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

## 1.6 The Bibliography

### 1.6.1 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 notes at the end of the chapter, paper, or book.

All entries in the bibliography are of the form '⟨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 in 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, ....

**Figure 1.7**: A minimal bibliography example.

[**Lamport, 1994**] L. Lamport. LaTeX: A Document Preparation System. Addison-Wesley, 1994.
[**Knuth, 1990**] D. E. Knuth. The TeXbook. Addison-Wesley, 1990. The source of this book is freely available from `http://www.ctan.org/tex-archive/systems/knuth/tex/`.

In LaTeX 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 as 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 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.[4] 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.

LaTeX Usage
```
\bibliographystyle{named}
\usepackage{named}
```

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

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

[4] But we should think about the environmental effects of using more paper than necessary.

**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].

**Figure 1.9**: The `\cite` command 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].

**`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 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 BibTeX program. This is explained further on.

The reason for putting the two braces (`{}`) after the `\LaTeX` and `\TeX` commands is technical. The following explains this in more detail. In LaTeX a group is treated as a word. However, LaTeX ignores spaces after most commands, including `\TeX` and `\LaTeX`. Without the empty groups, there would not have been proper inter-word spaceing between 'LaTeX' and 'package' and between 'TeX' and 'program' in the final output. However, adding the empty group 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 LaTeX 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[0]{\refname}{⟨other name⟩}`**

This redefines the command `\refname` to `⟨other name⟩`. The `\renewcommand` may also be used to redefine other exisiting commands. It is explained in Chapter 10.

**`\nocite{⟨list⟩}`**

This produces no text but writes the entries in the comma-separated list `⟨list⟩` 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.

### 1.6.2 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 startig 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 LATEX. The following explains the process.

1. You generate 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 LATEX 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:

   Unix Usage
   ```
   $ bibtex ⟨document⟩
   ```

   Here `⟨document⟩` is the basename of your top-level LATEX 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 LATEX's bibbliography. A common mistake of `bibtex` users is that they add the extension to the basename of the LATEX source file. It is not clear why this is not allowed.

**Figure 1.10**: Including a bibliography.

```
\documentclass[11pt]{article}
% Use bibliography style named.
% Requires the file named.bst.
\bibliographystyle{named}
% Use package style.
% 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}
```

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 removed, and when (3) you change the BIBTEX 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 LATEX source in this example depends on a BIBTEX file called '`db.bib`'.

The following is an example of two entries in a BIBTEX database file. The example associates the logical label '`Lamport:94`' with Lamport's LATEX book and the logical label '`Strunk:White:1979`' with the book about elements of style. The author, title, year of publication, 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.

BIBTEX

```
@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},
}
```

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 `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`.

### 1.6.3 The `natbib` Package

There are several problems with the basic LATEX citation mechanism. The `natbib` package overcomes some of them. It also provides a more flexible citation mechanism.[5]

- The `natbib` package distinguishes between *parenthetical* and *textual* citations. A parenthetical citation is similar to the default LATEX citation. With `natbib` you get such citations with the `\citep` command. A textual citation is used as the subject of a sentence. With `natbib` you get such citations with the `\citet` command. Figure 1.11 demonstrates how to use the commands.

[5] Unfortunately, I haven't been able to get the `natbib` package to work with the `beamer` class.

**Figure 1.11**: The \citet and \citep commands.

```
\citet{Knuth:1990}
 describes \TeX.
The ultimate guide to \TeX{}
 is~\citep{Knuth:1990}.
```

Knuth (1990) describes TeX. The ultimate guide to TeX is (Knuth, 1990).

**Figure 1.12**: The \citeauthor and \citeyear commands.

```
\citeauthor{Knuth:1990}
 wrote~\cite{Knuth:1990}
 in~\citeyear{Knuth:1990}.
```

Knuth wrote (Knuth, 1990) in 1990.

- The package also provides the commands \citeauthor and \citeyear which give you the author(s) and year of a citation. Figure 1.12 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.

LaTeX Usage

```
\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.
- ....

If you have a TeX Live installation then you can get information about the natbib package by executing the following command.

Unix Usage

```
$ texdoc natbib
```

Getting help for other packages and classes works similar. The natbib package may be downloaded from the Comprehensive TeX Archive Network (CTAN) at http://www.ctan.org. If you are looking for other classes and packages then CTAN is also the place to be.

### 1.6.4 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 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}
```
LaTeX Usage

3. Using \bibliographystyle — 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}
```
LaTeX Usage

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

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

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.
```
LaTeX Usage

5. You use the redefined \bibliography command:

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

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 BIBTEX database file. The previous example, uses the same 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}
```
LaTeX Usage

We're done with the work at the LaTeX level. This time we use BibTeX and apply it to the names of the bibliographies.

```
Unix Usage
$ bibtex books
$ bibtex articles
```

### 1.6.5 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**

## 1.7 Reference Lists

### 1.7.1 Table of Contents and Lists of Things

In this section you will learn 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. LaTeX also lets you define other reference lists. The following example, which should be easy enough to understand, shows us how to include a table of contents, and lists of figures and tables.

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

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 (odd-numbered) page, producing a blank page if necessary.

### 1.7.2 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

**Table 1.1**: Depth values of sectional unit commands.

| Sectional Unit Command | `\tocdepth` | `\secnumdepth` |
|---|---|---|
| `\part` | -1 | 1 |
| `\chapter` | 0 | 2 |
| `\section` | 1 | 3 |
| `\subsection` | 2 | 4 |
| `\subsubsection` | 3 | 5 |
| `\paragraph` | 4 | 6 |
| `\subparagraph` | 5 | 7 |

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.

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⟩}`.

### 1.7.3 Controlling the Sectional Unit Numbering

The counter `\secnumdepth` is related to the counter `\tocdepth`. Its value indicated the depth of the the sectional units that are numbered. So by setting the counter `\secnumdepth` to 3, you tell 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`.

### 1.7.4 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, LaTeX's default index mechanism only allows you to have one single index. This is why we shall use the `multind` package, which allows you to have 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`.

LaTeX Usage
```
\makeindex{programs}
\makeindex{authors}
```

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

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

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 simpel example which creates an entry 'TeX' in the index for the programs. (More information about the `\index` command may be found in [Lamport, 1994, Appendix A].)

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

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.

Unix Session
```
$ makeindex authors
$ makeindex programs
```

## 1.8 Class Files

As explained before, each top-level LATEX document corresponds to a document class. The document class is determined by the required argument of `\documentclass` command in your LATEX document.

LATEX Usage
```
\documentclass{⟨document class name⟩}
```

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 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.

**Figure 1.13**: A 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}
```

**`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 1.13 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 LaTeX indicate hyphenation and justification problems by putting a little square in the margin of the problem line.

**`final`:** Used for the final version.

## 1.9 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 Usage

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 package called `counter`.

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

To find out about the options you have to read the documentation. The `texdoc` utility helps you locate and read the documentation. If you have a `TeX-Live` or `miktex` LATEX installation the utility should be installed. The following shows how to use it.

```
$ texdoc style
```
Unix Usage

This kicks off a program which displays the documentation (provided it exists).

## 1.10 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.[6]

**`listings`:** Including program listings.

**`url`:** Typesetting URLs.

**`prettyref`:** Consistent typesetting cross-references.

**`amsmath`:** Typesetting tools from the American Mathematical Society (AMS).

**`fourier`:** Sets the text font to *Utopia Regular* and the math font to *Fourier*.

**`graphicx`:** Including graphics.

**`coverpage`:** User-defined coverpages.

**`fancyhdr`:** User-defined headers and footers.

[6] If you have other favourites then please let me know.

**`lastpage`:** Defines a command for getting the last page number. This is especially useful for $M/N$ page numbers.

**`memoir`:** This class provides support for writing books. The class comes with lots and lots of options for finetuning the typesetting.

**`keyval`:**

**`xkeyval`:** Parsing "`var=val`" style argument lists. Supports parsing of `key=val` lists.

**`classicthesis`:** Nice package for thesis.

**`mathtools`:** More precise typesetting of math.

## 1.11 Errors and Troubleshooting

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

# Part II

# Basic Typesetting

CHAPTER 2

# Running Text

This chapter studies the ingredients for writing text and changing the appearance of text. Section 2.1 shows how to write characters which are special to LaTeX, Diacritics are also studied in Section 2.1. Sections 2.3–2.11 cover ligatures, dashes, emphasis, footnotes and marginal notes, quotations, changing the size of the text, changing the type style of the text, alignment, the `booktabs` package, and phantom text. This is followed by Section 2.12, which shows how to align text. Section 2.13 concludes this chapter with language related issues.

## 2.1 Special Characters

This section studies ten characters which have a special meaning to LaTeX. It explains how to use the characters — they're operators really — and how to typeset them. Table 2.1 depicts the characters, their meaning, and the command to typeset them. We have already studied the start-of-comment operator (`%`) and the backslash (`\`) symbol, which starts control sequences. The command `\backslash` is only used when specifying mathematical formulae. It is described in Chapter 8. The command argument reference operator is described in Chapter 10. The alignment tab (`&`) is described in Section 2.12.3. The math mode switch operator (`$`), the subscript operator (`_`), and the superscript operator (`^`) are described in Chapter 8. The three remaining operators are described in the remainder of this section.

### 2.1.1 Tieing Text

It is recalled that 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 LaTeX (TeX really) determines the line breaks. The purpose of the tilde operator (`~`) is to specify interword space which should not 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.

**Table 2.1**: Ten special characters.

| Character | Purpose | Command |
|---|---|---|
| # | Command argument reference | `\#` |
| $ | Math mode switch operator | `\$` |
| % | Start of comment | `\%` |
| & | Vertical alignment tab | `\&` |
| ~ | Text tie operator | `\textasciitilde` |
| _ | Math subscript operator | `\_` |
| ^ | Math superscript operator | `\textasciicircum` |
| { | Start of group | `\{` |
| } | End of group | `\}` |
| \ | Start of control sequence | `\textbackslash` or `\backslash` |

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 `$`⟨math⟩`$` is used to typeset ⟨math⟩ 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 $f( x )$ is (1)~continuous; (2)~bounded`.

### 2.1.2 Grouping

Grouping is a commonly used technique in TeX and LaTeX. The left brace operator (`{`) starts a group. The right brace operator (`}`) closes it. Grouping has two purposes:

- The first advantage 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: it results in 'A bold **word** and a bold **l**etter.'.

  LaTeX Usage
  ```
  A bold \textbf{word} and a bold \textbf letter.
  ```

- The second advantage 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.

  LaTeX Usage
  ```
  Normal text here.
  { % Start a group.
    % Switch to bold face text.
    \bfseries % Now we have bold face text.
    Bold paragraphs in here.
  } % Close the group.
  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 (check) |
| ŏ | `\u{o}` | Breve accent |
| ō | `\={o}` | Macron (bar) |
| ő | `\H{o}` | Long Hungarian umlaut |
| o͡o | `\t{oo}` | Tie-after accent |
| ş | `\c{s}` | Cedilla accent |
| ọ | `\d{o}` | Dot-under accent |
| o̱ | `\b{o}` | Bar-under accent |

> Inside the group you may have several paragraphs. The advantage of the declaration `\bfseries` is that it allows you to change how the text is typeset for the rest of the entire group, which may have several paragraphs. With the `\textbf` command paragraph-breaks in the argument are not allowed.

There is also a low-level TEX operator pair for creating groups. It works just as the braces: only the operators are different. A group is started with `\begingroup` and ended with `\endgroup`. These operators may be freely mixed with braces but pairs should be properly matched. So '`{ \begingroup \endgroup }`' is allowed but '`{ \begingroup } \endgroup`' is not.

## 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 containing diacritics and the commands which typeset them in LATEX. The presentation is based on [Knuth, 1990, Chapter 9].

A common error is to use `\"{i}` to typeset ï. The proper way to typeset ï is using '`\"{\i}`'. 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.

## 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), ﬃ (ffi), and ﬄ (ffl). Fonts may also have a ligature for 'ff', but for some reason the font which is used in this document doesn't

**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 ‘es-zet’ or sharp S |

have one. LaTeX recognises English ligatures and will substitute them for the character combinations which they represent.

Table 2.4 displays some foreign ligatures. The 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 ‘fz’. Looking at the shape of the ligature, this sort of makes sense.

Sometimes you may prefer *not* to have ligatures. The following prevents LaTeX from turning the ‘ff’ in ‘shelfful’ into a ligature, which makes the result much easier to parse.

LaTeX Usage
```
He bought a shelf{}ful of books.
```

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

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

## 2.4 Quotation Marks

This section briefly explains how to typeset quotation marks. The following example is based on [Lamport, 1994, Page 13].

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

This results in the following output.

**Figure 2.1**: Quotation marks.

```
``\,`Fi' or `fum?'\,'' he asked.\\
```Fi' or `fum?''' he asked.\\
``{}`Fi' or `fum?'{}'' he asked.
```

" 'Fi' or 'fum?' " he asked.
"'Fi' or 'fum?"' he asked.
"'Fi' or 'fum?'" he asked.

Quotation marks. The LaTeX input to the left results in the output to the right.

'Convention' dictates that punctuation go inside quotes, like "this," but some think it's better to do "this". LaTeX Output

In the example the word 'Convention' is quoted using single quotes. The word 'this' is quoted using double quotes. The quotes at the start are backquotes (` and ``). The quotes at the end are the usual quotes (' and ''). Notice that the quote between 'it' and 's' is produced using a single quote in LaTeX.

Sometimes you need quoted text inside a quotation. To do this properly you should insert a *thin space* where quotes "meet". You can typeset a thin space with the command '\,'. Figure 2.1 provides a concrete example which is taken from [Lamport, 1994, Page 14].

The first line of the output in Fig 2.1 looks better. Note that LaTeX parses the sequence ''' in the second line of the input as a pair of quotes followed by one more quote. The last line of the input tries to overcome this by making explicit where the quotes 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 [Trask, 1997, Chapter 8]. This poses a problem with the single quote which is used for the posessive form: He said 'It is John's book.'. This is why it is also acceptable to use double quotes.*

## 2.5 Dashes

There are three kinds of dashes: -, –, and —. You can produce them in LaTeX using '-', '--', and '---'. The second symbol can also be typeset with the command \textendash and the last symbol with the command \textemdash. 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 LaTeX you typeset this symbol as follows: -.

**–:** This is the *en-dash*, which roughly has a width of the letter N. It is used in ranges: Pages 12–15. The LaTeX command \textendash and the sequence -- typeset the en-dash.

**—:** 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.

The following is an example.
```

LaTeX Usage
```
The intra-word dash is used to hyphenate
 compound modifiers such as light-green,
 X-ray, or one-to-one. ...
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}. ...
```

I recently found that sometimes --- doesn't work in `xelatex`. Using `\textemdash` fixes the problem.

## 2.6 Emphasis

*Emphasis* is an important tool when writing documents. LaTeX supports a declaration, a command, and an environment for emphasis.

The first of the three is the declaration `\em`. The best way to regard it is as an emphasis switch. The semantics of the declaration depend on how many times it has been used.

- If, at the current position in the block, the number of (active) declarations is even — the most common case — then the next declaration will switch to a slanted style.
- Otherwise, the current style is slanted, and the next declaration will switch back to the style which was active just before the first use of `\em` in the block.

The declarations are local to the group in which they are contained. The text style which is the result of the last `\em` declaration in a group remains valid until the end of the group.

LaTeX Usage
```
Text: {\em First, \em second, \em and third\/} text.
```

This results in the following.

LaTeX Output

Text: *First,* second, *and third* text.

The sequence '`\/`' in the example inserts an *italic correction* at the end of the group. This results in some extra space, which compensates for the fact that the last character in the group is slanted to the right. There is no need to insert italic corrections before punctuation marks with low heights. If you use it before other punctuation marks and letters the result looks better. The following LaTeX and the resulting output should demonstrate the difference.

LaTeX Usage
```
{\em This emphasised f\/} looks good.\\
{\em This emphasised f} looks bad.
```

This results in the following.

LaTeX Output

*This emphasised f* looks good.
*This emphasised f*look bad.

**Figure 2.2**: 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 of them, the more annoying they get.%
 \footnote{Got it?}
```

Footnotes[a] can be a nuisance. This is especially true if there are many of them.[b] The more you see of them, the more annoying they get.[c]

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

The second of the emphasis-related notions is `\emph`. It takes one argument. It works just as `\em` but this time the effect is local to the argument. The output of `\emph` is almost similar to the output which you get with `\em`. It appears that the italic correction is not required if you use the command `\emph`. There is also a command called `\textem`, which appears to be deprecated. The following example results in '*A short* example *with* nested *emphasis*.'.

LaTeX Usage

```
\emph{A short
\emph{example
\emph{with}
      nested
\emph{emphasis}}}.
```

The final of the emphasis-related notions is the `emph` environment. It appears that italic corrections are not needed if you use it.

## 2.7 Footnotes and Marginal Notes

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

Not surprisingly, LaTeX provides commands for footnotes and marginal notes. Figure 2.2 demonstrates how to specify footnotes in LaTeX.

In documents with small margins it is better to avoid marginal notes.

A *marginal note* or *marginal pargagraph* 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 and even pages: the optional argument is used for even pages and the default argument is used for odd pages. Note that marginal notes may look better with ragged text.

**Figure 2.3**: The `quote` environment.

```
Blah blah 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 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 blah blah.

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

| Declaration | Environment | Example |
|---|---|---|
| `\tiny` | `tiny` | Example |
| `\scriptsize` | `scriptsize` | Example |
| `\footnotesize` | `footnotesize` | Example |
| `\normalsize` | `normalsize` | Example |
| `\large` | `large` | Example |
| `\Large` | `Large` | Example |
| `\LARGE` | `LARGE` | Example |
| `\huge` | `huge` | Example |
| `\Huge` | `Huge` | Example |

## 2.8 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 2.3 provides an example which demonstrates how to use the `quote` environment.

The `verse` environment is for typesetting poetry and verse. The following demonstrates how to use it. In this example, the command `\qquad` is used to insert an amount of space which is equivalent to twice the width of the letter 'M'.

## 2.9 Controlling the Size

Changing the type size explicitly is usually not recommended in a LaTeX document. However, sometimes it has its merits, e.g. when you're designing your own titlepage or environment.

Table 2.5 lists the declarations which allow you to change the type size. The preferred "size" for

**Figure 2.4**: 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;        \\
    \qquad When they asked, ”Does it hurt?” \\
    \qquad 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.”

algorithms and program listings is `\scriptsize`.

The following provides a small example.

LaTeX Usage

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

It results in the following output.

LaTeX Output

Mumble.
What?
I said ‘Mumble’.

## 2.10 Controlling the Type Style

There are ten LaTeX declarations which affect the type style. For each declaration there is a corresponding command which allows you to apply the type same style (as you get with the declaration) to an argument. The declarations and commands are listed inTable 2.6.

**Table 2.6**: Type style affecting declarations and commands.

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

**Figure 2.5**: The `\phantom` command.

```
Fill in the missing word.\\
Fill in the missing
 \phantom{word}.
```

Fill in the missing word.
Fill in the missing       .

## 2.11 Phantom Text

The `\phantom` command "typesets" its argument using invisible ink. Figure 2.5 demonstrates how it is used.

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 when getting the right size for delimiters such as parentheses in mathematical formulae that span multiple lines. This is explained in Section 8.8.1.

## 2.12 Alignment

This section studies five methods which change the way the text is aligned. The first method lets you centre text. The second and third method are for aligning text to the left and to the right. The fourth method is the `tabular` environment. This method is useful for defining row-based content with columns occurring at regular/predictable positions. The final method is the `tabbing` environment. It provides a relatively high-level environment which allows you define vertical alignment (tab) positions in columns and lets you position text relative to these alignment positions.

### 2.12.1 Centred Text

Centering text is done with the center environment. The following is an example.

LaTeX Usage
```
\begin{center}
   Blah.\\
   Blah blah.\\
   Blah blah blah.

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

LaTeX Output

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

### 2.12.2 Flushed/Ragged Text

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

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

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

LaTeX Output

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.

### 2.12.3 The tabular Environment

The tabular environment is for specifying aligned text consisting of 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. The tabular* environment is not explained here. More information about this environment may be found in [Lamport, 1994, Appendix C.10.2].

It is recalled that LaTeX has a text mode and a math mode. The explanation of the array environment, which can only be used in math mode, is postponed until Chapter 8. The tabular and tabular* environments can be used in text and math mode.

The remainder of this section describes how to use the tabular environment. Readers interested in a complete explanation may find one in [Lamport, 1994, Appendix C.10.2].

**\begin{tabular}[⟨pos⟩]{⟨cols⟩} ⟨rows⟩ \end{tabular}**

The following explains the arguments of the environment:

**⟨pos⟩:** Specifies the vertical positioning in the rows. Allowed values are 't' (align to top), 'c' (align to the centre), or 'b' (align to the bottom). As the square bracket notation suggests, the "square-bracket argument" is optional. It defaults to c.

**⟨cols⟩:** Specifies alignment of the columns. This specification is a sequence of column options, where each column option is one of the following.[1]

- **l:** Column is left aligned.
- **r:** Column is right-aligned.
- **c:** The alignment in the column is centred.
- **p⟨width⟩:** A top-aligned column having width ⟨width⟩ which is typeset as a paragraphs in "the usual way."
- **@{⟨text⟩}:** Inserts ⟨text⟩ at corresponding position.
- **|:** Draws a vertical line. This is discouraged.

The last two options do not specify columns but decoration for the corresponding margins of the columns. For example, '`r@{---}l`' specifies that there should be an em-dash at the position between the right- and left-justified columns.

**⟨rows⟩:** Specifies the rows.

Now that we know how to specify the column alignment/decoration it is time to study how to specify the ⟨rows⟩. The ⟨rows⟩ is a sequence of ⟨row⟩ members, which are separated using the newline operator (`\\`):

```
⟨row⟩ \\  ⟨row⟩ \\ ... \\  ⟨row⟩
```
LaTeX Usage

This typesets each ⟨row⟩ as a row of the environment. Each ⟨row⟩ is a sequence of ⟨item⟩ specifications. Each ⟨item⟩ is aligned according to the corresponding specification in ⟨cols⟩. It is possible to override the default alignment; this is explained further on. The number of ⟨item⟩ specifications should not exceed the number of columns in ⟨cols⟩. The '&' symbol is used as an ⟨item⟩ separator. The items are typset with vertical alignment. The `tabular` environment wraps each ⟨item⟩ in a group. An individual ⟨item⟩ may contain:

**⟨text⟩**
This text is typeset in the current column according to the corresponding alignment in ⟨cols⟩.

**\multicolumn{⟨number⟩}{⟨col⟩}{⟨item⟩}**
This typesets ⟨item⟩ but uses ⟨number⟩ columns.

- Aligns ⟨item⟩ according to ⟨col⟩, which should contain exactly one of the symbols '`l`', '`c`', '`r`', or '`p`'. In addition it is also allowed to insert pipe symbols (`|`) which result in vertical grid lines drawn at the corresponding position. For example, using '`||c`' results in two vertical lines before the corresponding column, which is aligned to the centre.
- ⟨item⟩ may contain the command `\vline`. This draws a vertical line from the top to the bottom of the current row in the environment.

***{⟨number⟩}{⟨cols⟩}**
This is equivalent to ⟨number⟩ copies of ⟨cols⟩. Here ⟨number⟩ should be a positive integer and ⟨cols⟩ a list of column specifiers.

The following commands, which draw horizontal lines, may be used at the start of a ⟨row⟩.

[1] This description is not exhaustive.

`\hline`
Draws a horizontal line across full length of environment.

`\cline{⟨number⟩₁-⟨number⟩₂}`
Draws a horizontal line across Columns ⟨number⟩$_1$–⟨number⟩$_2$. This is useful for multi-column table headings.

The following is a not particularly meaningful example. The command `\eurologo`, which is used in the example, typesets the Euro symbol (€). The command is provided by the `fourier` package. (The `marvosym` package also provides a command for the Euro symbol. It is called `\EUR`.)

LaTeX Usage

```
\begin{tabular}[c]{lr@{ }lr@{.}lp{30mm}}
  \hline \multicolumn{1}{c}{\textbf{Destination}}
       & \multicolumn{2}{r}{\textbf{Duration}}
       & \multicolumn{2}{r}{\textbf{Price}}
       & \multicolumn{1}{r}{\textbf{Description}}
\\\hline Algarve
       &  1 & Week  & \eurologo  321 & 34 % Price
       & Blah blah blah blah blah blah blah blah.
\\       Gran Canaria
       & 12 & Weeks & \eurologo 4300 & 00
       & Blah blah blah blah blah blah blah blah.
\\\hline
\end{tabular}
```

The source results in the following output, which has been centred for clarity.

LaTeX Output

| **Destination** | **Duration** | **Price** | **Description** |
|---|---|---|---|
| Algarve | 1 Week | €321.34 | Blah blah blah blah blah blah blah blah. |
| Gran Canaria | 12 Weeks | €4300.00 | Blah blah blah blah blah blah blah blah. |

The '|' option and the `\cline`, `\vline`, and `\hline` commands are irresistible to new users. However, using them in moderation is better since grid lines makes scanning the contents of the table difficult. Chapter 6 provides some guidelines on how to design tables.

### 2.12.4 The booktabs Package

The `booktabs` package adds some extra functionality to the `tabular` environment. The package discourages the use of 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.

**Figure 2.6**: Using booktabs.

```
\begin{tabular}[c]{lr@{ }lr@{.}lp{30mm}}
  \toprule \multicolumn{1}{l}{\textbf{Destination}}
         & \multicolumn{1}{r}{\textbf{Duration}}
         & \multicolumn{2}{r}{\textbf{Price}}
         & \multicolumn{1}{r}{\textbf{Description}}
\\\midrule Algarve
         &  1 & Week  & \eurologo  321 & 34
         & Blah blah blah blah blah blah blah blah.
\\       Gran Canaria
         & 12 & Weeks & \eurologo 4300 & 00
         & Blah blah blah blah blah blah blah blah.
\\\bottomrule
\end{tabular}
```

- 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 commands which are provided by booktabs are as follows.

**\toprule[⟨width⟩]**
Used for the first horizontal rule in the table. The optional argument is the width of the rule.

**\bottomrule[⟨width⟩]**
Used for the last horizontal rule in the table. The optional argument is the width of the rule.

**\midrule[⟨width⟩]**
Used for the remaining full horizontal rules in the table. The optional argument is the width of the rule.

**\cmidrule[⟨width⟩]{⟨number⟩$_1$-⟨number⟩$_2$}**
Used for the remaining a partial horizontal rule through Columns ⟨number⟩$_1$–⟨number⟩$_2$. The optional argument is the width of the rule.

**\addlinespace[⟨width⟩]**
This command is usually used immediately after a line break and (guess) gives you more vertical line space.

Figure 2.6 shows how to use these commands. The output is listed in Figure 2.7. Notice that the inter-linespacing is much better than the spacing in the table on Page 48. Also notice the different widths of the rules.

**Figure 2.7**: Output of the input listed in Figure 2.6.

| Destination | Duration | Price | Description |
|---|---|---|---|
| Algarve | 1 Week | €321.34 | Blah blah blah blah blah blah blah blah. |
| Gran Canaria | 12 Weeks | €4300.00 | Blah blah blah blah blah blah blah blah. |

### 2.12.5 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*.

**`\=`**
: Defines the next tab (vertical alignment) position.

**`\\`**
: Inserts line break and resets next tab position to `left_margin_tab`.

**`\kill`**
: Throws away the current line but remembers the tab positions defined with `\=`.

**`\+`**
: Increments `left_margin_tab`.

**`\-`**
: Decrements `left_margin_tab`.

**`\>`**
: Move to the next tab stop.

Figures 2.8–2.8 present two examples of how to use the `tabbing` environment. The examples do not demonstrate the full functionality of the environment.

## 2.13 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.

**Figure 2.8**: The `tabbing` environment.

```
\begin{tabbing}
From \=here to \=there \\
   \>and \>then\\
   \>\>all\\
   \>the \>way\\
back \=to \=here.
\end{tabbing}
```

```
From here to there
     and    then
            all
     the    way
back to     here.
```

**Figure 2.9**: Advanced use of `tabbing` environment.

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

```
FUNC euc( INT a, INT b ) :  INT
BEGIN
  WHILE (b != 0) DO
  BEGIN
    INT rem = a MOD b;
    a = b;
    b = rem;
  END
  RETURN a;
END;
```

### 2.13.1 Hyphenation

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

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

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

- A cleaner solution is to use the `\hyphenation` command in the preamble. The 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.

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

### 2.13.2 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. The following provides a minimal example.

LaTeX Usage
```
\usepackage[dutch,british]{babel}
⋮

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

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

### 2.13.3 Spelling

LaTeX does not support automatic text spelling. The `vim` program has a spell-checker plugin which is called `SpellChecker`. The `ispell` program supports LaTeX. The '`-t`' flag tells the command that the input is LaTeX.

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

CHAPTER

3

# Lists

In this chapter we shall study environments for *lists*. In Section 3.1 we shall study unordered lists. This is followed by Section 3.2, which studies ordered lists. Section 3.3 describes the `enumerate` package, which provides a high-level interface to control the labels which are used for the lists. In Section 3.4 we shall study 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*, which are lists the order of whose items is irrelevant-ish. In the body of the environment you start each item in the list using the `\item` command. There should be at least one `\item`. It is possible to have nested itemized lists. Figure 3.1 demonstrates how to use the `itemize` environment.

Each item is marked with a *label*. Usually, the top-level label is a black circle but the default appearance of the labels may depend on the class and style files which you are using. The command `\labelitemi` defines the default appearance for the label of the top-level items, `\labelitemii` for the labels of the subitems, `\labelitemiii` for the labels of the subsubitems, and `\labelitemiv` for the labels of the subsubsubitems. By redefining these commands, you can change the appearance of the labels. You may redefine an existing command using the `\renewcommand` command. The following shows how to use a plus sign for the top level item label and a minus sign for the fourth level item label. The command `\renewcommand` is discussed in more detail in Chapter 10.

LaTeX Usage

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

If you only want to change the appearance of a given label for a single list then the easiest way to do this is to keep the redefinition of the `\labelitemi` command and the list in a *group*. You may create a group using braces. See Section 2.1.2 for further information about the merits of groups. The following shows how to change the appearance of the label at Level 1.

**Figure 3.1**: The `itemize` environment.

```
\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 nested 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 nested item.

**Figure 3.2**: The `enumerate` environment.

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

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

LaTeX Usage

```
{ % Start a group.
  % Definitions are local to the group.
  \renewcommand{\labelitemi}{+}
  \begin{itemize}
     \item Labelled with a plus sign.
  \end{itemize}
} % End of group.

\begin{itemize}
   \item Labelled with the default label.
\end{itemize}
```

## 3.2 Ordered Lists

The `enumerate` environment is for creating ordered lists. It works just as the `itemize` environment but this time the labels of the items are labelled with numbers, letters, or roman numerals. Figure 3.2 demonstrates how to use 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 commands \labelenumi, \labelenumii, \labelenumiii, and \labelenumiv. Each of these commands depends on a *counter* which is used to count the number of times the \item command has been used at the corresponding level. Counters are explained 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. The commands \arabic, \roman, \Roman, \alph, and \Alph are used to typeset a given counter using arabic numbers, lower case roman numerals, upper case roman numerals, lower case letters, and upper case letters. The following demonstrates how to redefine the command for labelling the top level items with lower case roman numerals, the command for labelling the second level items with upper case letters, and the command for labelling the third level items with numbers.

LaTeX Usage
```
\renewcommand{\labelenumi}{\roman{enumi}}
\renewcommand{\labelenumii}{\Alph{enumii}}
\renewcommand{\labelenumiii}{\arabic{enumiii}}
```

## 3.3 The enumerate Package

The enumerate package proves a high-level interface to LaTeX'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 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. The following is an example.

LaTeX Usage
```
\usepackage{enumerate}
⋮
% Start enumerated list with labels A-A, A-B, A-C, ...
\begin{enumerate}[{A}-A]
\item First item.
\item Second item.
...
\end{enumerate}
```

The interested reader is referred to the package documentation [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 3.3 provides an example of how to use the description environment.

**Figure 3.3**: The `description` environment.

```
Kurasawa films include:
\begin{description}
\item[Kagemusha (1980):]
     When a powerful warlord
     in medieval Japan dies,
     a poor thief is recruited
     to impersonate him. ...
\item[Yojimbo (1961):]
     A crafty ronin comes
     to a town divided by
     two criminal gangs. ...
\item[Sanshiro Sugata (1943):]
     A young man struggles to
     learn the nuance and
     meaning of judo. ...
\end{description}
```

Kurasawa films include:

**Kagemusha (1980):** When a powerful warlord in medieval Japan dies, a poor thief is recruited to impersonate him. ...

**Yojimbo (1961):** A crafty ronin comes to a town divided by two criminal gangs. ...

**Sanshiro Sugata (1943):** A young man, struggles to learn the nuance and meaning of judo. ...

## 3.5 Making your Own Lists

LaTeX's `list` environment is for defining your own lists. It works as follows.

**`\begin{list}{⟨label commands⟩}{⟨formatting commands⟩} ⟨item list⟩ \end{list}`:**
The `⟨label commands⟩` consists of commands that typeset the labels. For ordered lists you may need to define a dedicated counter that keeps track of the numbers of the labels. The `⟨formatting commands⟩` consists of commands that format the resulting list. The formatting depends on *length* commands. Figure 3.4 depicts how the lengths determine the formatting of the list. The picture is based on [Lamport, 1994, Figure 6.3]. The vertical dimensions correspond to rubber lengths. The horizontal dimensions correspond to rigid lengths.

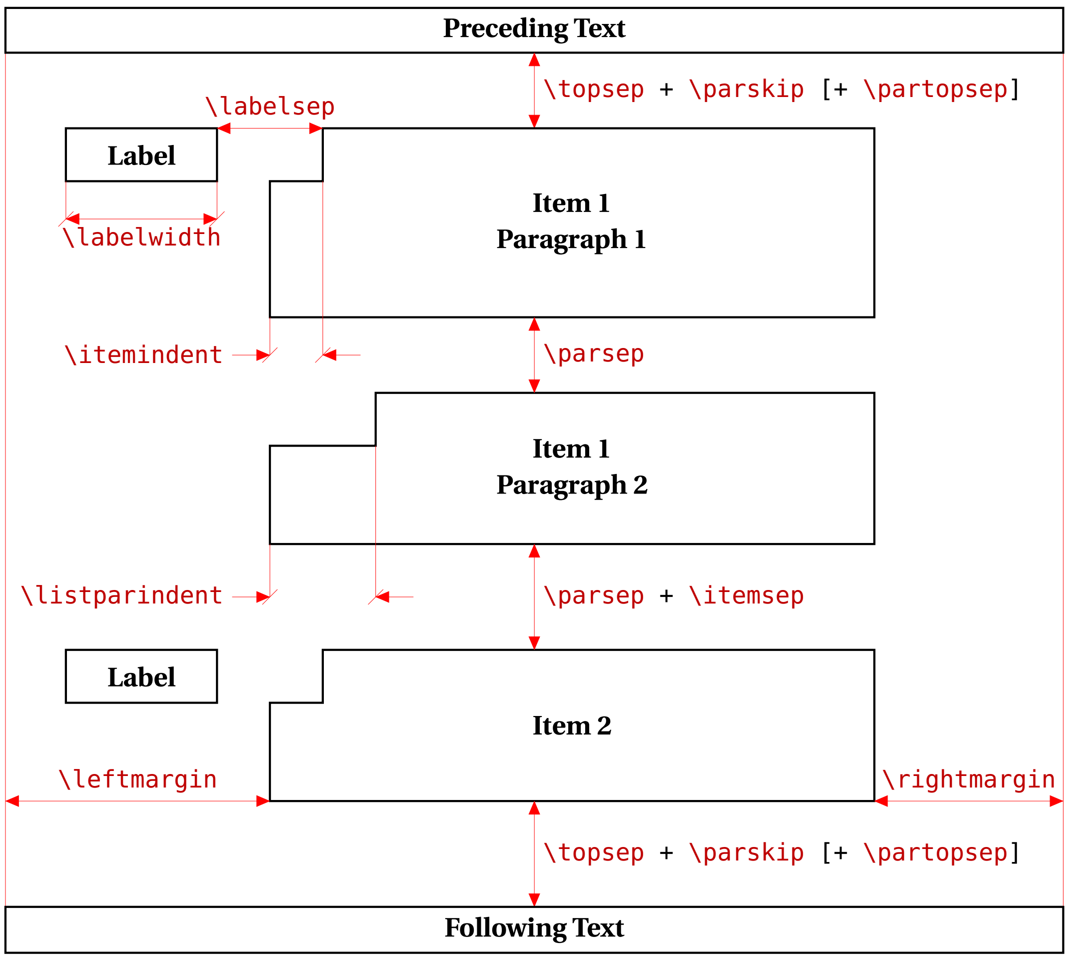
The following is an example.

LaTeX Usage
```
\newcounter{ListCounter}

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

The command `\newcounter{ListCounter}` defines the counter `ListCounter`. The spell '`List-\alph{ListCounter}`' typesets the label of each item as '`List-`' followed by the current value of the counter as a lower case letter. Inside the second argument of the `list` environment, the command

**Figure 3.4**: Lengths which affect the formatting of a LaTeX `list` environment.



\usecounter{ListCounter} "uses" the counter, which basically adjusts the value of the counter for the next \item.

As part of 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. This lets us re-implement the previous example as follows.

LaTeX Usage

```
\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}
```

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.

The remainder of this chapter, is mainly based on [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 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* figure. Here *floating* refers to the fact that the actual figure may be typeset at a different location than where the figure is defined. This mechanism gives LaTeX some freedom to choose better page breaks for the remaining text. For example, if there's not enough room left for the figure at the "current" position then LaTeX may fill up the remainder of the page with more paragraphs and put the picture on the next page. The resulting figure may end up in a different place in the output document than where you'd expect it. The result is an easthetically more pleasing document.

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

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 mechanism 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`', and '`h`', which are used as follows [Lamport, 1994, Page 197]:

**t:** At the top of a page.

**b:** At the bottom of a page.

**p:** On a separate page containing no text, but only figures, tables, and other floats.

**h:** 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'. 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. An optional argument may be 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 any LaTeX statements to produce the actual picture.

LaTeX Usage

```
\begin{figure}[tbp]
   ⟨Insert any LaTeX code to produce the picture.⟩
   \caption[Comparison of algorithms.]
           {Comparison of algorithms. ...
            \label{fig:comparison}}
\end{figure}
```

## 4.2 External Picture Files

A common mechanism for creating pictures is including them from external files. The best picture formats are vector graphics formats: `.eps`, `.pdf`, .... 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 LaTeX are `.eps` and `.pdf`.

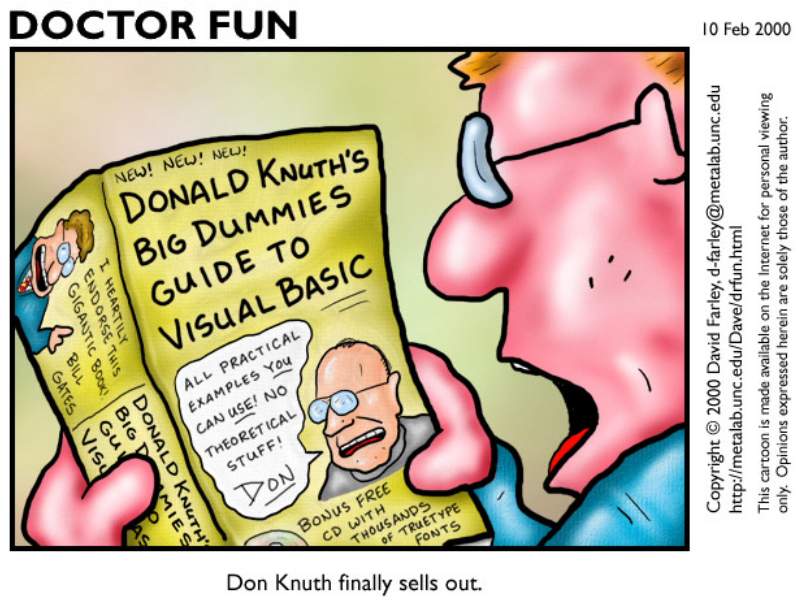
Programs such as `gnuplot` may be used to generate graphs in vector graphics format. A common practice is to generate complicated graphs with `gnuplot` and then include them with LaTeX. This mechanism is relatively easy. However, `gnuplot` may not always have the right graph output style. Another problem 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.)

## 4.3 The `graphicx` Package

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

**Figure 4.1**: Including an external graphics file.

```
\begin{figure*}[tbp]
\centering
\includegraphics
    [width=110mm]
    {vb4dummies.jpg}
\end{figure*}
```




Including an external graphics file with the `includegraphics` command. The input to the left results in the output to the right.

**`\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 4.1 shows an example of the `\includegraphcs` 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`).

## 4.4 Setting Default Key Values

The `graphicx` package uses the `keyval` package to handle its `⟨key⟩=⟨value⟩` pairs. The `keyval` packages lets you define a default value for each key. This is done with the `\setkeys` command. Without going into any details of the `keyval` package, which is explained in Section 11.2, the command \setkeys{Gin}{⟨list⟩} sets the default values for the keys. 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

## 4.5 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

## 4.6 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{}.

## 4.7 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`.

Unix Script

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

## 4.8 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 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 are using pdfLaTeX then the following extensions are supported: `.png`, `.pdf`, `.jpg`, and `.mps`. Here the `.mps` format is for METAPOST source files, which are programs 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 [Carlisle, 2003].

CHAPTER

5

# Presenting Diagrams with tikz

This chapter is an introduction to drawing diagrams 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 TEX and LATEX. As a result tikz also lets you integrate text and mathematics in your diagrams. The tikz package also supports the beamer package, which is used for creating incremental presentations. (Chapter 18 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 TEX and LATEX, so graphical objects also can be text. What is more, the things which are specified/drawn 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 computations 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 environment.

Each tikzpicture results in a box. The size of the box is the smallest possible box containing the typeset material. 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 $2\times2$ square which is drawn at coordinate $(0,0)$ in your tikzpicture. All *implicit* units inside a tikzpicture are in centimetres. Scaling a picture 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: .

LaTeX 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 picture. 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}`'.

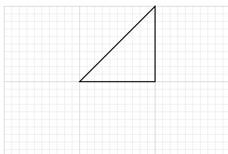
**\tikz[⟨options⟩] ⟨command⟩;**

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 the things in a picture. Grids are also useful when you are developing a picture. The following are two ways to draw a grid. The former way is easier, but it is expressed in terms of the second, more general, notation.

**Figure 5.1**: Drawing a grid.



```
\begin{tikzpicture}
\draw[line width=0.1pt,gray!20,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}
```

**\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.

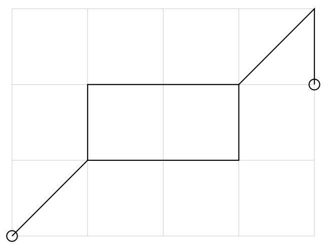
The option, style really, 'help lines' is very useful as it results in lines drawn in a subdued colour. For the online version of this book the style is redefined to make the lines very thin and to set the color to a combination of 20% black and 80% white. This was done with the command `\tikzset{help lines/.style={very thin,color=black!20}}`. 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 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:

**Figure 5.2**: Creating a path.



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (4,3);
\draw (0,0) circle (2pt)
        -- (1,1) rectangle (3,2)
        -- (4,3)
        -- (4,2) circle (2pt);
\end{tikzpicture}
```

LaTeX Input

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

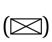
The first `\path` command in this `tikzpicture` draws a line segment from (1,0) to (2,0). The second `\path` command results in a line segment which is not drawn. Still the line segment is considered part of the picture, so the picture has a width of 3 cm.

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 which are entirely defined 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 '`(⟨label⟩)`' as if it were the coordinate. The following, which draws a crossed rectangle (), demonstrates the mechanism. It is not intended to excel in terms of maintainability.

LaTeX Input

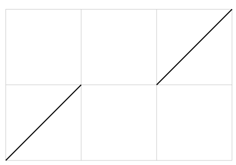
```
The following, which draws a crossed rectangle
(\begin{tikzpicture}
 \draw (0.0,0.0) coordinate(lower left)
         -- (0.4,0.2) coordinate(upper right);
 \draw (0.0,0.2) -- (0.4,0.0);
 \draw (lower left) rectangle (upper right);
 \end{tikzpicture}), demonstrates the mechanism.
```

## 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 ... ⟨coordinate⟩ ... ;**

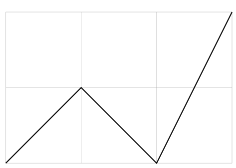
This is the *move-to* operation, which simply adds the coordinate ⟨coordinate⟩ to the path. The following example uses three move-to operations. The first move-to operation defines the lower left corner of the grid. The remaining move-to operations define the starts of two line segments.



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

**\path ... -- ⟨coordinate⟩ ...;**

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 ⟨coordinate⟩. The following is an example.



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

**\path ... .. controls ⟨coordinate$_1$⟩ and ⟨coordinate$_2$⟩ .. ⟨coordinate$_3$⟩ ...;**

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 ⟨coordinate$_3$⟩, and the control points are ⟨coordinate$_1$⟩ and ⟨coordinate$_2$⟩.

Figure 5.3 demonstrates the operation. The curve starts at `c1` and ends at `c4`. Its control points are given by `c2` and `c3`. The tangent of the spline segment at `c1` is given by the tangent of the line segment '`c1 -- c2`'. Likewise, the tangent of the spline segment at `c4` is given by the tangent of the line segment '`c3 -- c4`'. 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.

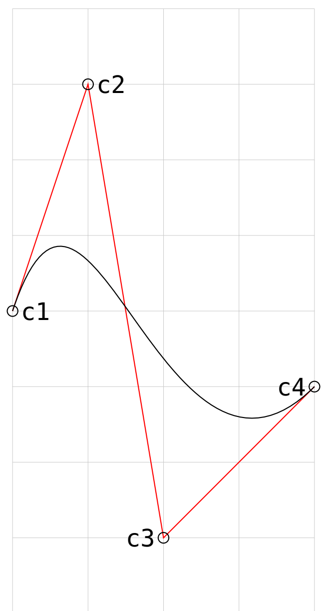
**\path ... .. controls ⟨coordinate$_1$⟩ .. ⟨coordinate$_2$⟩ ...;**

This is also a curve-to operation. It is equivalent to the curve-to operation '... `.. controls` ⟨coordinate$_1$⟩ `and` ⟨coordinate$_1$⟩ `..` ⟨coordinate$_2$⟩ ...'.

**\path ... -- cycle ...;**

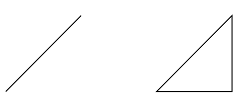
This is the *cycle* operation which *closes* the current path by adding a straight line segment

**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}
```

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 want 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.

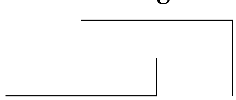


LaTeX Input

```
\tikz \draw (0,0) -- (1,1)
            (2,0) -- (3,0) -- (3,1) -- cycle;
```

**`\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.

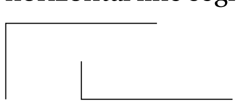


LaTeX Input

```
\tikz \draw (0.0,0.0) -| (2.0,0.5)
            (1.0,1.0) -| (3.0,0.0);
```

**`\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.

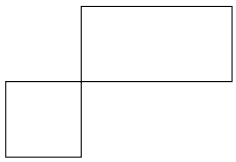


LaTeX Input

```
\tikz \draw (0.0,0.0) |- (2.0,1.0)
            (1.0,0.5) |- (3.0,0.0);
```

**`\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.

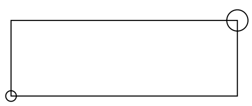


LaTeX Input

```
\begin{tikzpicture}
\draw (0,0) rectangle (1,1)
            rectangle (3,2);
\end{tikzpicture}
```

**`\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.

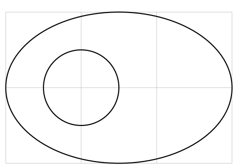


LaTeX Input

```
\tikz \draw (0,0) circle (2pt)
                  rectangle (3,1)
                  circle (4pt);
```

**`\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.

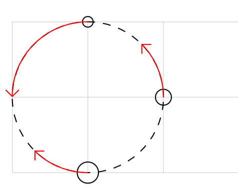


LaTeX Input

```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (3,2);
\draw (1.0,1.0) ellipse (0.5cm and 0.5cm)
      (1.5,1.0) ellipse (1.5cm and 1.0cm);
\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 exampe, 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.



LaTeX Input

```
\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.

**Table 5.1**: Available xcolor colours.

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

## 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]`, which draws the following path.

**`\fill`**
This is a shorthand for `\path[fill]`, which fills the following path.

**`\filldraw`**
This is a shorthand for `\path[filldraw]`, which fills and draws the following path.

**`\shade`**
This is a shorthand for `\path[shade]`, which shades the following path.

**`\shadedraw`**
This is a shorthand for `\path[shadedraw]`, which shades and draws the following path.

### 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 ⟨ratio⟩ should be a real in the interval $[0:1]$.

**\colorlet{⟨name⟩}{⟨colour⟩!⟨percentage⟩}**

This defines a new colour called ⟨name⟩ which is the result of mixing ⟨percentage⟩% ⟨colour⟩ and (100 − ⟨percentage⟩)% white. Here ⟨colour⟩ should be the name of an existing colour.

**\colorlet{⟨name⟩}{⟨colour$_1$⟩!⟨percentage⟩!⟨colour$_2$⟩}**

This defines a new colour called ⟨name⟩ which is the result of mixing ⟨percentage⟩% ⟨colour$_1$⟩ and (100−⟨percentage⟩)% ⟨colour$_2$⟩. Here ⟨colour$_1$⟩ and ⟨colour$_2$⟩ 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 allow you to set the colour which is used 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 which is used for drawing and filling. It also sets the colour of the text in nodes.

You can set the colour of the whole `tikzpicture` or set the colour of a given path action. The former is done by passing a '`color=`⟨colour⟩' option to the environment, the latter 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.

LaTeX Usage
```
\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 `tikz` 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.

LaTeX Usage
```
\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 draw in red. The other is drawn in blue.

LaTeX Usage
```
\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⟩.

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

**ultra thin**

This sets the line width to 0.1 pt.



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

**very thin**

This sets the line width to 0.2 pt.



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

**thin**

This sets the line width to 0.4 pt.



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

**semithick**

This sets the line width to 0.6 pt.



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

**thick**

This sets the line width to 0.8 pt.



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

**very thick**

This sets the line width to 1.2 pt.

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

**ultra thick**

This sets the line width to 1.6 pt.

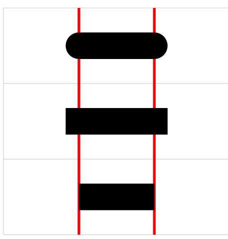
LaTeX Input
```
\tikz \draw[ultra thick] (0,0) -- (2,4pt);
```

### 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’.



LaTeX Input

```
\begin{tikzpicture}[line width=10pt]
\draw[help lines] (0,0) grid (3,3);
\draw[line width=1pt,red] (1,0) -- (1,3)
                          (2,0) -- (2,3);
\draw[line cap=round] (1,2.5) -- (2,2.5);
\draw[line cap=rect]  (1,1.5) -- (2,1.5);
\draw[line cap=butt]  (1,0.5) -- (2,0.5);
\end{tikzpicture}
```

**line join=⟨style⟩**

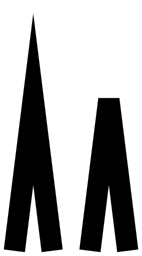
This sets the line join style to ⟨style⟩. There are three possible values for ⟨style⟩: ‘round’, ‘bevel’, and ‘miter’.

LaTeX Input

```
\begin{tikzpicture}[line width=8pt]
\draw[line join=round] (0.0,.8) -- (0.3,.0) -- (0.6,.8);
\draw[line join=miter] (0.9,.0) -- (1.2,.8) -- (1.5,.0);
\draw[line join=bevel] (1.8,.8) -- (2.1,.0) -- (2.4,.8);
\begin{tikzpicture}
```

**miter limit=⟨fraction⟩**

This option is useful only if you’re using the miter line join style with sharp angles, which may result in the miter join protruding too far beyond the joining point. The length which is given by the product of ⟨fraction⟩ and the line width is used as a limit on how far the miter join is allowed to protrude the joining point. In the event of the join protruding beyond the limit, the join style is changed to bevel.



LaTeX Input

```
\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);
\begin{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 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). The following is an example which uses millimetres for simplicity.

LaTeX Input
```
\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}
```

**dash phase=⟨dimension⟩**

This shifts the dash phase by ⟨dimension⟩.

LaTeX Input
```
\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}
```

**solid**

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

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

**dotted**

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

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

**densely dotted**

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

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

**loosely dotted**

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

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

**dashed**

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

LaTeX Input
```
\tikz \draw[dashed] (0,0) -- (2,0);
```

**densely dashed**

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

LaTeX Input
```
\tikz \draw[densely dashed] (0,0) -- (2,0);
```

**loosely dashed**

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

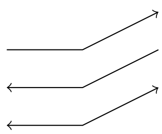
LaTeX Input
```
\tikz \draw[loosely dashed] (0,0) -- (2,0);
```

### 5.8.6 Arrows

Arrows are also drawn using path actions.

**arrows=⟨arrow head$_1$⟩-⟨arrow head$_2$⟩**

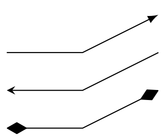
This adds an arrow head to the start and to the end of the path. You may also omit the '`arrows=`' and use the shorthand notation '⟨arrow head$_1$⟩-⟨arrow head$_2$⟩'. The arrow head at the start is determined by ⟨arrow head$_1$⟩. The arrow head at the end is determined by ⟨arrow head$_2$⟩. Omiting ⟨arrow head$_1$⟩ omits the arrow head at the start of the path. Omitting ⟨arrow head$_2$⟩ 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`.



LaTeX Input
```
\begin{tikzpicture}
\draw[->]  (0,1.0) -- (1,1.0) -- (2,1.5);
\draw[<-]  (0,0.5) -- (1,0.5) -- (2,1.0);
\draw[<->] (0,0.0) -- (1,0.0) -- (2,0.5);
\end{tikzpicture}
```

**>=⟨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 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 put in the preamble of your document. The following provides a small example.



LaTeX Input
```
\begin{tikzpicture}
\draw[>=latex,->]     (0,1.0) -- (1,1.0) -- (2,1.5);
\draw[>=stealth,<-]  (0,0.5) -- (1,0.5) -- (2,1.0);
\draw[>=diamond,<->] (0,0.0) -- (1,0.0) -- (2,0.5);
\end{tikzpicture}
```

### 5.8.7 Filling a Path

Not only can paths be draw they can also be filled with a colour or filled with a colour and drawn with another colour. The only requirement is that the path be closed. Clossing 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`.

LaTeX Input
```
\begin{tikzpicture}[scale=0.4,fill=cyan]
\path[fill]            (0,0) rectangle (1,1);
\path[fill=red]        (2,0) -- (3,0) -- (3,1) -- cycle;
\path[fill,color=blue] (4,0) -- (5,0) -- (5,1) -- ;
\end{tikzpicture}
```

**Table 5.2**: Arrow head types.

| | | | | | |
|---|---|---|---|---|---|
| | | | | | **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 | | * | |

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.

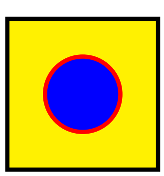
**\fill[⟨options⟩] ⟨paths⟩;**

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 pased to \path and the paths in ⟨paths⟩ are filled.

LaTeX Input
```
\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⟩;**

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

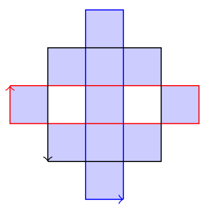


LaTeX Input
```
\begin{tikzpicture}[scale=0.5]
\filldraw[ultra thick,fill=yellow]
         (0,0) rectangle (4,4);
\filldraw[ultra thick,fill=blue,draw=red]
         (2,2) circle (1cm);
\end{tikzpicture}
```

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 Nonzero Rule** The option 'nonzero rule' is the default. A point is considered inside a collection of paths if it is enclosed by a different number of paths which are drawn in clockwise direction

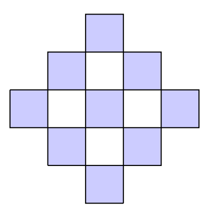
**Figure 5.4**: The ‘nonzero rule’.



```
\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}
```

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

**Figure 5.5**: The ‘even odd rule’.



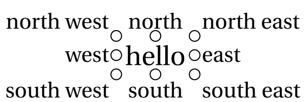
```
\begin{tikzpicture}[fill=blue!20,scale=0.5]
\fill[even odd rule]
      (0,2) -- (0,3) -- (5,3) -- (5,2)
      (2,0) -- (3,0) -- (3,5) -- (2,5)
      (1,1) -- (4,1) -- (4,4) -- (1,4);
\draw (0,3) -- (5,3) -- (5,2) -- (0,2) -- (0,3);
\draw (3,0) -- (3,5) -- (2,5) -- (2,0) -- (3,0);
\draw (1,1) -- (4,1)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

Using the ‘even odd rule’ to determine which area is filled. The fill involves three rectangles. For the ‘even odd rule’ a point is inside the shape (is filled) if it requires the crossing of an odd number of lines to get from the point to “infinity”.

than paths which are drawn counter-clockwise. To complicate matters closed paths may “overlap” themselves and this may result in points which are in clockwise as well as counter-clockwise 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 considered inside the paths if the number of times $\ell$ crosses a clockwise drawn line differs from the number of times $\ell$ crosses an anti-clockwise line. Figure 5.4 depicts an example. This example does *not* have self-overlapping paths.

**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 5.5 depicts an example. This example also does not have self-overlapping paths.

**Figure 5.6**: 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.
```

## 5.9 Nodes and Node Labels

Pictures consisting of only lines are rare. Usually, you want your diagrams to contain some text or math. Fortunately, `tikz` has a mechanism to add 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 `⟨content⟩` at the current position in the path using the options `⟨options⟩` and associates the label `⟨label⟩` with the node. The outer shape of the node is only drawn if '`draw`' is part of `⟨options⟩`. The default shape is a rectangle but other shapes are also defined. The following section explains how to control the node shape. The texts '`(⟨label⟩)`' and '`[⟨options⟩]`' are optional.

**`\draw ... node(⟨label⟩)[⟨options⟩]{⟨content⟩} ... ;`**

This is similar to the `\path` version before.

For example, the command '`\draw (0,0) node {hello};`' draws the word 'hello' at the origin. Likewise, the following draws a circle and the word 'circle' at position (1,0).

LaTeX Input

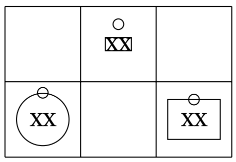
```
\draw (0,1)          % make (0,1) current position.
      circle (2pt)   % draw circle at current position.
      node {circle}; % draw word circle at current position.
```

When a node gets a label, `⟨label⟩`, then usually the additional labels '`⟨label⟩.center`', '`⟨label⟩.north`', '`⟨label⟩.north east`', ..., and '`⟨label⟩.north west`' are also defined. The coordinates of these labels correspond to their names, so '`⟨label⟩.north`' is to the north of the node having label `⟨label⟩`. This holds for the most common node shapes. Figure 5.6 provides an example which involves all these auxiliary labels, except for '`⟨label⟩.center`'.

### 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 manymore 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.

**Figure 5.7**: 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 shape of a node is determined by the '`shape` = ⟨`shape`⟩' option. The following are the basic predefined node shapes:

**`coordinate`:** This shape is for coordinates. Coordinates have no shape and they cannot be drawn.

**`rectangle`:** This shape is for rectangular nodes. The rectangle is fit around ⟨`content`⟩. This is the default option.

**`circle`:** This shape is for circles. The circle is fit around ⟨`content`⟩.

**`ellipse`:** This shape is for ellipses. The ellipse is fit around ⟨`content`⟩.

The default height and width of a node are not always ideal. Fortunately, there are options for low-level control. The minimum and maximum width, height, and size of a node are controlled with the options '`minimum width` = ⟨`dimension`⟩', '`maximum width` = ⟨`dimension`⟩', '`minimum height` = ⟨`dimension`⟩', '`maximum height` = ⟨`dimension`⟩', '`minimum size` = ⟨`dimension`⟩', and, finally, '`maximum size` = ⟨`dimension`⟩'. All these options work as "expected".

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 of ⟨`content`⟩ and the border.

There are also options '`inner xsep` = ⟨`dimension`⟩', '`outer xsep` = ⟨`dimension`⟩', '`inner ysep` = ⟨`dimension`⟩', and '`outer ysep` = ⟨`dimension`⟩' controlling the separations in the horizontal and vertical directions. They work "as expected".

Figure 5.7 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 *anchor* 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.

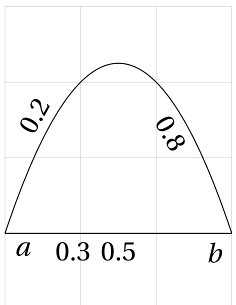
**`rotate=⟨angle⟩`**
: Draws the node, but rotates it `⟨angle⟩` degrees about its anchor point.

**`pos=⟨ratio⟩`**
: This option is for placing nodes along a path (as opposed to at the current coordinate). Here `⟨ratio⟩` is non-negative real less than or equal to 1. This option places the node at the relative position on the path which is determined by `⟨ratio⟩`, so if `⟨ratio⟩` 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.

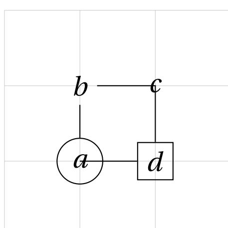
**`midway`**
: This option is equivalent to using '`pos = 0.5`'. Likewise, the option '`start`' is equivalent to

**Figure 5.8**: 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 5.9**: 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}
```

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 5.8 shows an example of some of these `node` options. Notice that several nodes can be placed with `pos` options for the same path segment.

### 5.9.3 Connecting Nodes

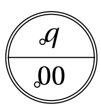
The `tikz` package is well behaved. It won't cross lines unless you say so. This includes the crossing of borderlines of node shapes. For example, let's assume you've created two nodes. One of them is a circle, which is labelled ⟨c⟩, and the other is a rectangle, which is labelled ⟨r⟩. When you draw a line using the command '`\draw (⟨c⟩) -- (⟨r⟩);`' then the resulting line segment will *not* join the centres of the two nodes. The actual line segment will be shorter as 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 5.9 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` defines more shape styles. The following are some interesting shape styles.

**`circle split`**

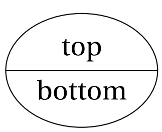
This defines a circular shape with an upper and a lower *node part*. The `node` option `double` may be used to draw the circle with a double line. The node parts of the `circle split` are separated by a line. The contents of the node parts are specified in the argument of the `\node`. The `\nodepart` command is used to delimit the different node parts of the `circle split`, so `\nodepart{⟨part⟩}` starts the nodepart `⟨part⟩`. The `circle split` shape allows two values for `⟨part⟩`. The first of these values is `text`. This value corresponds to the upper part of the `circle split`. The second allowed value is `lower`. It corresponds to the text in the lower part of the `circle split`. There is no need to explicitly call `\nodepart{text}` at the start of the node. In addition to the `circle` labels, the shape also defines a label for the `lower` part. The following is an example.




LaTeX Input
```
\tikz \draw (0,0)
        node(double)[circle split,draw,double]
                    {$q$ \nodepart{lower} $00$}
        (double.lower) circle (1pt)
        (double.text)  circle (1pt);
```

**`ellipse split`**

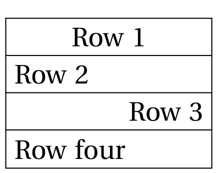
This is the ellipse version of `circle split`. The following is an example.




LaTeX Input
```
\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 following example 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.




LaTeX Input
```
\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};
```

## 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 way comes with its own specific *coordinate system*. Inside a coordinate system you specify coordinates using *explicit* or *implicit* notation.

**Explicit:** Explicit coordinates are specified by writing '`(`⟨system⟩ `cs:`⟨coord⟩`)`', where ⟨system⟩ is the name of the coordinate system and where ⟨coord⟩ is a coordinate whose syntax depends on ⟨system⟩. For example, to specify the point having $x$-coordinate ⟨x⟩ and $y$-coordinate ⟨y⟩ in the `canvas` coordinate system you write '`(canvas cs:x=`⟨x⟩`,x=`⟨y⟩`)`'.

**Implicit:** Implicit coordinates are specified inside parentheses in some syntax which is specific to the particular coordinate system. All 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 is used to specify coordinates relative to the origin on the paper. The implicit notation '`(`⟨x⟩`,`⟨y⟩`)`' specifies the point having $x$-coordinate ⟨x⟩ and $y$-coordinate ⟨y⟩.

**XYZ Coordinate System** The '`xyz`' coordinate system is used to define points which are expressed as a linear combination of three vectors, called the $x$-, $y$-, and $z$-vector respectively. 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 '`(`⟨x⟩`,` ⟨y⟩`,`⟨z⟩`)`' is used to define the point which is located at ⟨x⟩ times the $x$-vector plus ⟨y⟩ times the $y$-vector plus ⟨z⟩ times the $z$-vector.

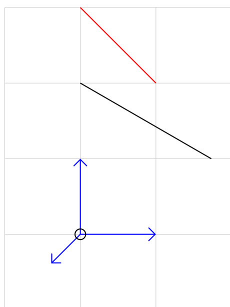
**Polar Coordinate System** The `canvas polar` coordinate system is used for specifying polar coordinates, i.e. coordinates which can be expressed by specifying an angle and a radius. The implicit notation '$(\alpha : r)$' corresponds to the point $r \times (\cos\alpha, \sin\alpha)$. Angles in this coordinate system, as all angles in `tikz`, should be supplied in degrees.

**Node Coordinate System** The '`node`' coordinate system uses labels to specify coordinates. The implicit notation '`(`⟨label⟩`)`' is the position of the node or coordinate which was given the label ⟨label⟩.

Figure 5.10 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`'.

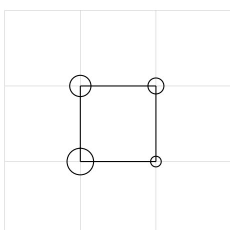
**Perpendicular Coordinate System** The '`perpendicular`' coordinate system is a dedicated system for computing intersections of horizontal and vertical lines. With this coordinate system's implicit syntax you write '`(`⟨$\text{pos}_1$⟩ `|-` ⟨$\text{pos}_2$⟩`)`' for the coordinate at the intersection of the infinite vertical line though ⟨$\text{pos}_1$⟩ and the infinite horizontal line through ⟨$\text{pos}_2$⟩. Likewise, '`(`⟨$\text{pos}_1$⟩ `-|` ⟨$\text{pos}_2$⟩`)`' results in the intersection of the infinite horizontal line though ⟨$\text{pos}_1$⟩ and the infinite vertical line through ⟨$\text{pos}_2$⟩. The notation for this coordinate system is quite suggestive as '`|`' suggests

**Figure 5.10**: 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 5.11**: Computing the intersection of perpendicular lines.



```
\begin{tikzpicture}
\draw[help lines] (0,0) grid +(3,3);
\path (1,1) coordinate (ll);
\path (2,2) coordinate (ur);
\draw (ll) -- (ll -| ur) circle (2pt);
\draw (ll -| ur) -- (ur) circle (3pt);
\draw (ur) -- (ur -| ll) circle (4pt);
\draw (ur -| ll) -- (ll) circle (5pt);
\end{tikzpicture}
```

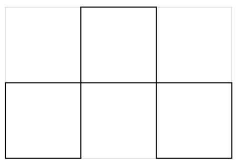
the vertical aspect of the line and '`-`' suggests the horizontal aspect of the other line. The order of the lines is then given by the order inside the operators '`|-`' and '`-|`'. Inside the parentheses you are not supposed to use parentheses for coordinates and labels, so you write '`(0,1 |- 1,2)`', '`(label |- 1,2)`', and so on. Figure 5.11 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 absolut coordinates requires changing $n$ coordinates. Fortunately, `tikz` has operations which compute coordinates from other coordinates. If used intelligently, these operations help you reduce 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.

**Figure 5.12**: 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}
```

### 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 allows you to construct 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 coordinate also lets you to construct a new coordinate at an offset from the current coordinate. This time, however, the current coordinate becomes the new coordinate. You use the implicit notation '++⟨offset⟩' for incremental coordinates.

Figure 5.12 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

Finally, `tikz` offers complex coordinate calculations. However, these calculations are only available if the `tikz` library `calc` is loaded in the preamble: `\usetikzlibrary{calc}`.

**([⟨options⟩]$⟨coordinate computation⟩$)**

This is the general syntax. The ⟨coordinate computation⟩ should:

1. Start with '⟨factor⟩∗⟨coordinate⟩⟨modifiers⟩'. Here ⟨modifiers⟩ is a sequence of one or more ⟨modifier⟩s and '⟨factor⟩∗' is an optional multiplication factor which defaults to 1. Both are described further on.
2. Continue with one or more expressions of the form: '⟨sign option⟩⟨factor⟩∗⟨coordinate⟩⟨modifiers⟩', where ⟨sign option⟩ is an optional '+' or '-'.

**⟨factor⟩**

Each ⟨factor⟩ is an optional numeric expression which is parsed by the `\pgfmathparse` command. Examples of valid ⟨factor⟩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 ⟨factor⟩s are optional and that the opening parenthesis 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.

**⟨modifier⟩**

A ⟨modifier⟩ is a postfix operator which acts on the coordinate preceding it. There are three different kinds: ⟨pmod⟩, ⟨dmod⟩, and ⟨prmod⟩. Each of them is of the form '!⟨stuff⟩' and it is used after a coordinate. To explain the modifiers we shall write ⟨partway modifier⟩ for ⟨coordinate⟩!⟨pmod⟩, shall write ⟨distance modifier⟩ for ⟨coordinate⟩!⟨dmod⟩, and shall write ⟨projection modifier⟩ for ⟨coordinate⟩!⟨prmod⟩.

**⟨partway modifier⟩**

There are two different forms of ⟨partway modifier⟩s. The first form is '⟨coordinate$_1$⟩!⟨factor⟩!⟨coordinate$_2$⟩' The resulting coordinate is given by

$$\langle\texttt{coordinate}_1\rangle + \langle\texttt{factor}\rangle \times (\langle\texttt{coordinate}_2\rangle - \langle\texttt{coordinate}_1\rangle)\,.$$

In words this is the coordinate which is at ⟨factor⟩/100% distance along the line between ⟨coordinate$_1$⟩ and ⟨coordinate$_2$⟩.

A ⟨partway modifier⟩ may also be of the more complex form '⟨coordinate$_1$⟩!⟨factor⟩!⟨angle⟩:⟨coordinate$_2$⟩'. The result is given by first computing ⟨coordinate$_1$⟩!⟨factor⟩!⟨coordinate$_2$⟩ and rotating the resulting coordinate about ⟨coordinate$_1$⟩ over ⟨angle⟩ degrees. Figure 5.13 presents an example of coordinate computations involving partway modifiers.

**⟨distance modifier⟩**

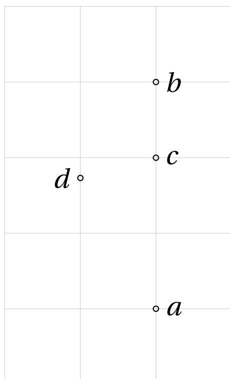
The next modifier is the ⟨distance modifier⟩. It is of the form '⟨coordinate$_1$⟩!⟨distance⟩:⟨angle⟩!⟨coordinate$_2$⟩', where ':⟨angle⟩' is optional. In its simpler form, '⟨coordinate$_1$⟩!⟨distance⟩!⟨coordinate$_2$⟩' results in the coordinate at distance ⟨distance⟩ in the direction from ⟨coordinate$_1$⟩ to ⟨coordinate$_2$⟩. For example, if the two coordinates are at distance 2 cm apart then setting ⟨distance⟩ to `1cm` gives you the point in between the two coordinates. The more complex form '⟨coordinate$_1$⟩!⟨distance⟩:⟨angle⟩!⟨coordinate$_2$⟩' works similar to the partway modifier. The resulting coordinate is obtained by first computing ⟨coordinate$_1$⟩!⟨distance⟩!⟨coordinate$_2$⟩ and rotating the result of this distance modifier expression about ⟨coordinate$_1$⟩ over ⟨angle⟩ degrees. Figure 5.14 presents an example of coordinate computations involving distance modifiers.

**⟨projection modifier⟩**

The final ⟨modifier⟩ is ⟨projection modifier⟩. This modifier is of the form '⟨coordinate$_1$⟩!⟨coordinate$_2$⟩!⟨coordinate$_3$⟩'[1] and it results in the projection of ⟨coordinate$_2$⟩ on the infinite line through ⟨coordinate$_1$⟩ and ⟨coordinate$_3$⟩. Figure 5.15 presents an example of coordinate computations with projection modifiers. (For some reason my `tikz` version doesn't like extra space around '!' inside a ⟨projection modifier⟩. It is not clear if this is a feature.)

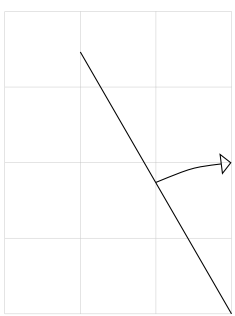
[1] The manual on Page 119 also mentions an angle but it is not explained how to use it....

**Figure 5.13**: Coordinate computations with partway modifiers.



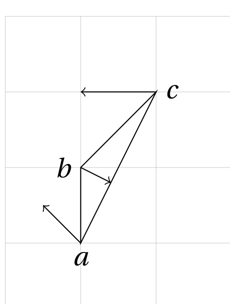

```
\begin{tikzpicture}
\draw[help lines] (0,0) grid +(3,5);
\draw (2.0,1.0)           circle (1pt)
                          coordinate(a)
                          node[anchor=west] {$a$}
      (2.0,4.0)           circle (1pt)
                          coordinate(b)
                          node[anchor=west] {$b$}
      ($(a)!0.666!(b)$)   circle (1pt)
                          node[anchor=west] {$c$}
      ($(a)!0.666!30:(b)$) circle (1pt)
                          node[anchor=east] {$d$};
\end{tikzpicture}
```

**Figure 5.14**: 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 5.15**: 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}
```

## 5.12 Options

Many tikz commands and environments depend on options. Usually these options are specified using '⟨key⟩ = ⟨value⟩' combinations. Some combinations have shorthand notations. For example, '⟨colour⟩' is a shorthand notation for '⟨color⟩ = ⟨colour⟩'. Options are best defined by passing their ⟨key⟩ = ⟨value⟩ combinations as part of the optional argument. However, there is another mechanism.

**\tikzset{⟨options⟩}**
Sets the options in ⟨options⟩. 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.

## 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 auxilliary 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.

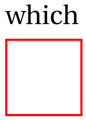
**Simplicity:** Changing the appearance of a graphical element with styles with well-understood interfaces is much easier and leads to fewer errors.

**Refinement:** You can stepwise refine the way certain graphics are draw. This allows you to postpone certain design decisions, but lets you start drawing 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 subgraphics.

**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, using 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 style which sets draw to red and uses a thick line.



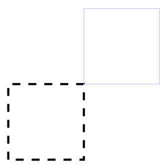
LaTeX Input

```
\tikzset{cork style/.style={draw=red,thick}}
\tikz \draw[cork style] (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⟩. The following is an example.



LaTeX Input

```
\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}
```

## 5.14 Scopes

Scopes in tikzpicture environments serve a similar purpose as blocks in a programming language and groups in LaTeX. 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.

**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  of by set the style using the options of a tikzpicture environment.

LaTeX Usage

```
\begin{tikzpicture}[every scope/.style={⟨list⟩}]
...
\end{tikzpicture}
```

The following options allow you to execute code at the start and end of the scope.

**execute at begin scope scope=⟨code⟩**

This option results in the execution of ⟨code⟩ at the start of the scope.

**execute at end scope scope=⟨code⟩**

This option results in the execution of ⟨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 5.16 depicts an example.

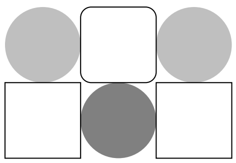
## 5.15 The \foreach Command

As if tikz productivity isn't enough, its pgffor 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

**Figure 5.16**: Using scopes.



```
\begin{tikzpicture}
\begin{scope}[fill=gray!50]
   \fill (0.5,1.5) circle (0.5);
   \begin{scope}[fill=gray]
      \fill (1.5,0.5) circle (0.5);
   \end{scope}
   \fill (2.5,1.5) circle (0.5);
\end{scope}
\draw (0,0) rectangle (1,1)
      { [rounded corners] rectangle (2,2) }
      (2,1) rectangle (3,0);
\end{tikzpicture}
```

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⟩. The following is an example.

4 3

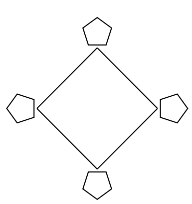
1 2

LaTeX Input

```
\tikz
\foreach \pos/\text in {{0,0}/1,{1,0}/2,{1,1}/3,{0,1}/4}
{
   \draw (\pos) node {\text};
}
```

Notice that the previous 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 it 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.



LaTeX Input

```
\tikz \draw (0,-0.8)
\foreach \angle in {0,90,180,270} {
   -- (\angle:0.8)
      (\angle:1.0) + (\angle:0.2)
   \foreach \fraction in {1,2,3,4,5} {
      -- +(\angle+\fraction*72:0.2)
   } -- cycle (\angle:0.8)
};
```

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.

**Table 5.3**: Shorthand notation for the `\foreach` command.

| Command | Yields |
|---|---|
| `\foreach \x in {1,2,...,6} {\x,}` | 1, 2, 3, 4, 5, 6, |
| `\foreach \x in {1,3,...,10} {\x,}` | 1, 3, 5, 7, 9, |
| `\foreach \x in {1,3,...,11} {\x,}` | 1, 3, 5, 7, 9, 11, |
| `\foreach \x in {1,0.1,...,0.5} {\x,}` | 0, 0.1, 0.20001, 0.30002, 0.40004, |
| `\foreach \x in {a,b,...,d,9,8,...,6} {\x,}` | a, b, c, d, 9, 8, 7, 6, |
| `\foreach \x in {7,5,...,0} {\x,}` | 7, 5, 3, 1, |
| `\foreach \x in {Z,X,...,M} {\x,}` | Z, X, V, T, R, P, N, |
| `\foreach \x in {1,...,5} {\x,}` | 1, 2, 3, 4, 5, |
| `\foreach \x in {5,...,1} {\x,}` | 5, 4, 3, 2, 1, |
| `\foreach \x in {a,...,e} {\x,}` | a, b, c, d, e, |
| `\foreach \x in {2^1,2^...,2^6} {$\x$,}` | $2^1$, $2^2$, $2^3$, $2^4$, $2^5$, $2^6$ |
| `\foreach \x in {0\pi,0.5\pi,...\pi,3\pi} {$\x$,}` | $0\pi$, $0.5\pi$, $1.5\pi$, $2.0\pi$, $2.5\pi$, $3.0\pi$ |
| `\foreach \x in {A_1,..._1,D_1} {$\x$,}` | $A_1$, $B_1$, $C_1$, $D_1$, |

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.

## 5.16 The let Operation

The let operation allows for the binding of expressions to “variables” inside a path. The following is the general syntax.

**\path … let ⟨assignments⟩ in …;**

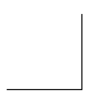
Here ⟨assignments⟩ is a comma-delimited list of assignments. Each assignment is of the form ‘⟨register⟩ = ⟨expression⟩’. To carry out the assignment, ⟨expression⟩ is evaluated and then assigned to ⟨register⟩, which is some variable which is local to tikz. After the assignments, the values of the variables may be got using the macros \n, \p, \x, and \y. However, this is only possible in the scope of the assignment which lasts 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⟩}. They are used to store numeric values. The second kind are point registers. These start with \p{⟨name⟩} and are used to store point/coordinate values. The following explains both ⟨register⟩s in more detail.

**\n{⟨name⟩} = ⟨expression⟩**

After this assignment the value which is assigned to the number register ⟨name⟩ may be got with the command \n{⟨name⟩}. The ⟨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.

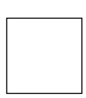
The following example demonstrates the use of number assignments.



LaTeX Input
```
\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⟩. A point which is assigned to the point register ⟨name⟩ may be got with the command \p{⟨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.



LaTeX Input
```
\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;
```

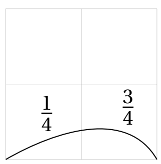
## 5.17 The To Path Operation

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 drawing paths with the `to` operation. This allows you to draw more 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. The following is an example.

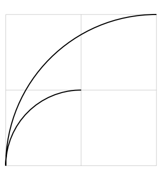



LaTeX Input
```
\begin{tikzpicture}
\draw[help lines] (0,0) grid (2,2);
\draw[out=30,in=120]
     (0,0) to node[pos=0.25,above] {$\frac{1}{4}$}
              node[pos=0.75,above] {$\frac{3}{4}$}
              (2,0);
\end{tikzpicture}
```

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.



LaTeX Input
```
\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⟩`**

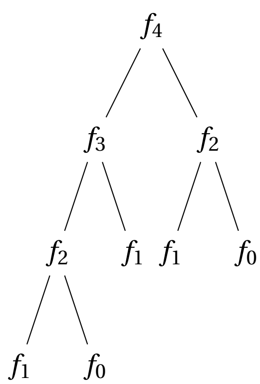
With this option, the following path is inserted: '`{[every to,⟨options⟩] ⟨path⟩}`'. Here ⟨options⟩ are the options passed to the `to` path. Inside ⟨path⟩ you can use the commands `\tikztostart`, `\tikztotarget`, and `\tikztonodes`. The value of `\tikztostart` is the start node of and that of `\tikztotarget` 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.

## 5.18 The spy Library

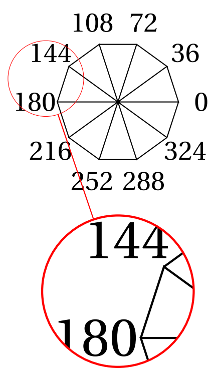
The `spy` library lets you magnify parts of diagrams. These magnifications are technically known as *canvas transformations*, which means they affect everything, including line widths, font size, and so on.

To use the feature you need to add the option '`spy scope`' to a picture or a scope to indicate it contains a part which is to be magnified. Some options implicitly add this option. I've noticed problems with the `spy` feature and `xelatex`. Fortunately it does work with `pdflatex`.

**Figure 5.17**: Drawing a tree.




```
\begin{tikzpicture}
      [level 2/.style={sibling distance=10mm}]
  \node {$f_4$}
        child {node {$f_3$}
               child {node {$f_2$}
                      child {node {$f_1$}}
                      child {node {$f_0$}}}
               child {node {$f_1$}}}
        child {node {$f_2$}
               child {node {$f_1$}}
               child {node {$f_0$}}};
\end{tikzpicture}
```




LaTeX Input

```
\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}
```

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].

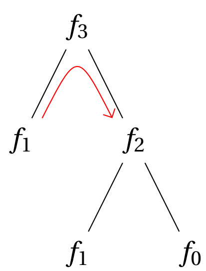
## 5.19 Trees

Knowing how to define node labels and knowing how to draw nodes and basic shapes, we are ready to draw some more interesting objects. 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. By default, the children of a parent are drawn from left to right in order of appearance. Unfortunately, drawing trees with tikz isn't perfect. The 'sibling distance = ⟨dimension⟩' 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 5.17 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 as well. Given label ⟨parent⟩ for a parent node, its $i$th child is

**Figure 5.18**: Using implicit node labels in tree.




```
\begin{tikzpicture}
  \node (top) {$f_3$}
        child {node {$f_1$}}
        child {node {$f_2$}
               child {node {$f_1$}}
               child {node {$f_0$}}};
  \draw[red,-angle 90]
       (top-1.north east) .. controls (top.south)
                          .. (top-2.north west);
\end{tikzpicture}
```

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.

labelled ⟨parent⟩-$i$. This process is continued recursively, so the $j$th child of the $i$th child of our parent node is labelled ⟨parent⟩-$i$-$j$. Figure 5.18 demonstrates the mechanism.

Changing the node style is a piece of cake. Figure 5.19 provides an example which sets the styles of the second and third level to different defaults. The option '`level distance = ⟨dimension⟩`' sets the distance between the levels in the tree.

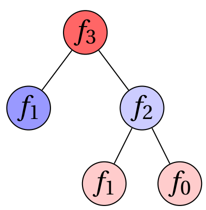
As already noted, the rules for automatic node placement are not always ideal. For example, sometimes you may wish to have the single child of a given parent drawn to the left or to the right of the parent. The `child` option `missing` allows you to specify a node which takes up space but which is not 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. By omitting a node, its label becomes inaccessible. Figure 5.20 provides an example.

## 5.20 Installing tikz

This temporary section describes how to obtain a recent `tikz`/`pgf` distribution and install it.

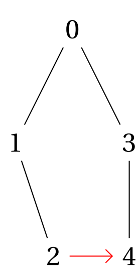
The easiest way to obtain a recent distribution is going to `http://sourceforge.net/projects/pgf` and saving a recent build. You install the distribution as a normal LaTeX package. This is best done by installing it locally in a dedicated directory called `${HOME}/${LaTeX}/${styles}/` (or equivalent) which hosts all your private LaTeX style and class files. Section 19.3 describes how this is done.

**Figure 5.19**: 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 5.20**: Missing tree nodes.




```
\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}
```

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.

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 LaTeX'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 LaTeX 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.

**Figure 6.1**: Components of a demonstration table.

| | **Table 3.1. GP and diabetic services, 2000** | | | |
|---|---|---|---|---|
| | | **GP Practices** | | |
| Column headings | **Towns** | **Number** | **Number providing diabetic services** | **% Providing diabetic services** |
| | Town A | 40 | 38 | 95 |
| | Town B | 29 | 27 | 93 |
| Row headings | Town C* | 29 | 25 | 86 |
| | Town D | 34 | 29 | 85 |
| | Town E | 36 | 30 | 83 |
| | Town F | 62 | 32 | 52 |
| | Total | 230 | 181 | 82 |
| Source | Source: Health Authority annual Report, 2001 | | | |
| Footnote | * Including Town E and Town F. | | | |

(The label "Number and Title" stands beside the table's title row.)

This table shows the components of a typical demonstration table. The background of the table is coloured grey. The black-on-white text to the left of the table describes the components of the table.

## 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.

**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 LaTeX, 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.

**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 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?
- 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.

**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 |

- The vertical alignment makes is difficult to compare the numbers in the table.
- 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 is 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 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.
- 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 reduces to the absolute minimum. The advantage is that it makes it easier to scan the data in the table.

  - 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 monospace 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: `$1.4 10^{+4}$` and `$2.3 10^{-3}$`. 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.

- 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.

The following shows how to create a table, which assumes the `booktabs` package is included.

LaTeX Usage

```
\begin{table}[tbp]
   \begin{tabular}{ll}
   \toprule
       \textbf{Chilled Meats}
         & \textbf{Calories per} \\
         & \textbf{100\,$\mathrm{g}$/4\,$\mathrm{oz}$}  \\
   \midrule
       ...
   \bottomrule
   \caption[Calories of chilled meats.]
           {Calories of chilled meats per weight. ...
            \label{tab:meat}}
\end{table}
```

## 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 ⟨`stuff`⟩ in a rotated table.

LaTeX Usage

```
\begin{sidewaystable}
   ⟨stuff⟩
\end{sidewaystable}
```

Inside ⟨`stuff`⟩, 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 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.

**Figure 6.2**: Using the `longtable` package.

```
\begin{longtable}{lr}
    \toprule
    \textbf{Meats}
      & \multicolumn{1}{l}{\textbf{Calories per $100\,\mathrm{g}$}}
  \\\midrule
\endfirsthead
    \toprule
    \multicolumn{2}{c}{\textbf{\tablename~\thetable\ Continued}}
  \\\midrule
    \textbf{Meats}
      & \multicolumn{1}{l}{\textbf{Calories per $100\,\mathrm{g}$}}
  \\\midrule
\endhead
    \midrule
    \multicolumn{2}{l}{\textbf{Continued on next page}}
  \\\bottomrule
\endfoot
    \bottomrule
\endlastfoot
    Salami        & 500
  \\Liver sausage & 300
  ⋮
\end{longtable}
```

**`\endfirsthead`**
: This indicates the end of the first column headings specification. All 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 `\endlatsfoot` is used to typeset the bottom of the table on the last page.

Figure 6.2 presents an example of LaTeX input which uses the `longtable` environment to typeset a table about the nutritional values of chilled meats.

## 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 packaged provide additional functionality which lets you create barcharts, piecharts, and so on. The following are some of these packages.[1]

**`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.

**`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.

[1] 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.

CHAPTER

7

# Presenting Data with Graphs

This chapter, which is still in its infancy, studies the presentation of data with graphs with 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

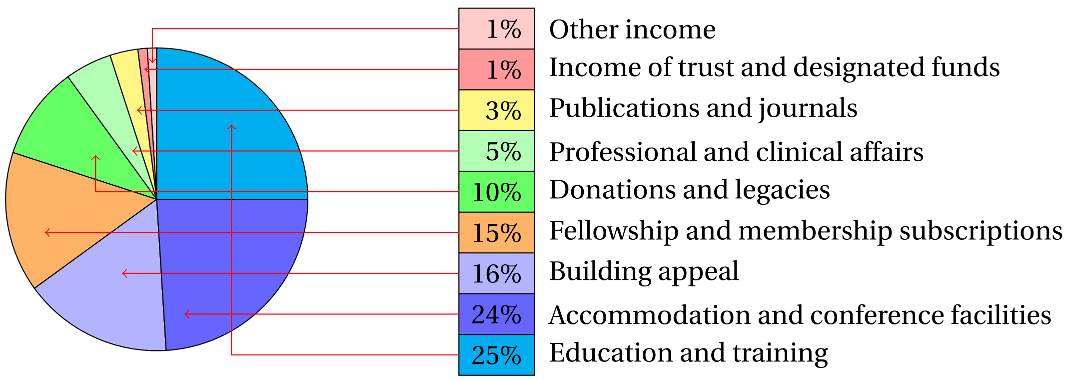
A picture can say more than a thousand words. This, to some extent, epitomises graphs: good graphs tell a story which is easily recognised and will stick. Graphs are good at showing global relationships, differences, and change.

**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.

**Figure 7.1**: A pie chart.




A well-designed graph sticks and conveys the essence of the relationship.

## 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.[1] 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 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. I'm not aware of any nice packages for drawing pie charts. So, in short, if after all the advise against pie charts you still insist on drawing a pie chart then you're on your own.

[1] 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.

## 7.3 Introduction to pgfplots

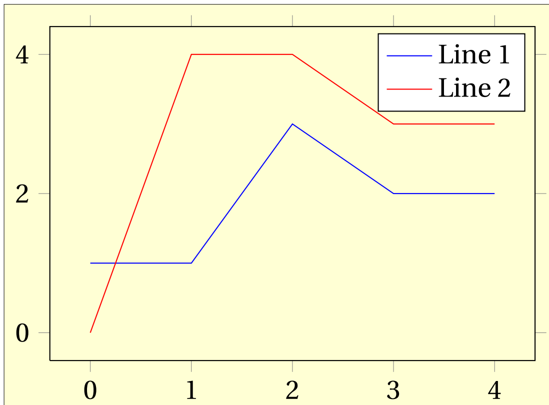
This section provides an introduction to drawing graphs with the pgfplots package which is built on top of pgf. The pgfplots package lets you draw graphs in a variety of formats. The resulting graphs have a consistent, professional look and feel. The package also lets you import data from matlab. As 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 draw 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. The following is a short example.

LaTeX Input
```
\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}
```

The following is the resulting output.

LaTeX 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.

LaTeX Input
```
\pgfplotsset{compat=newest,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 7.2 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.

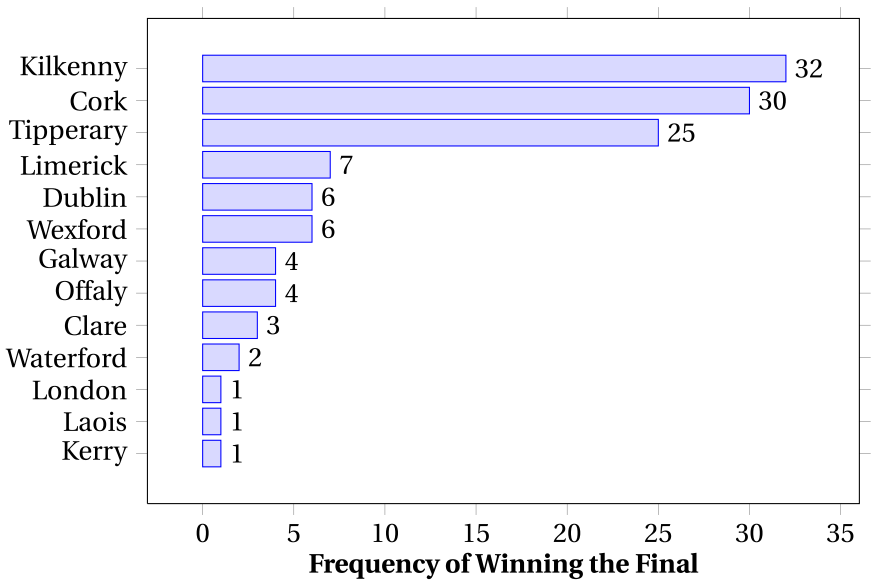
Figure 7.3 presents the input which was used to create the bar graph in Figure 7.2. The graph itself is typeset inside an `axis` environment which itself is inside a `tikzpicture` environment. The `xbar` option of the `axis` environment specifies that this is a horizontal bar graph. The data for the 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 7.3 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 7.2 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.

**Figure 7.2**: A bar graph.

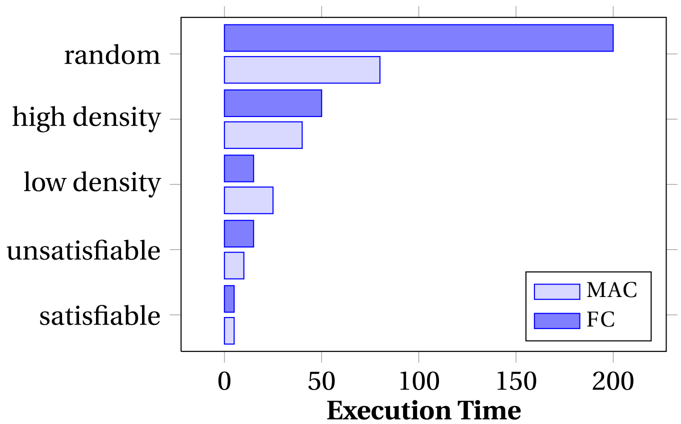



All-Ireland hurling champions and the number of times they've won the title before 2010.

**Figure 7.3**: Creating a bar graph.

```
\begin{tikzpicture}
\begin{axis}
      [xbar, width=11cm, height=8cm, bar width=10pt, enlargelimits=0.13,
       nodes near coords, nodes near coords 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) (25,11) (30,12) (32,13)};
\end{axis}
\end{tikzpicture}
```

**Figure 7.4**: A paired bar graph.




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 7.5**: 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}
```

## 7.5 Paired Bar Graphs

*Paired bar graphs* are like bar graphs but they present information about two groups of data. Figure 7.4 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 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 7.5 shows the input which was used to create the horizontal paired bar graph from Figure 7.4. 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.

## 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 7.6 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.

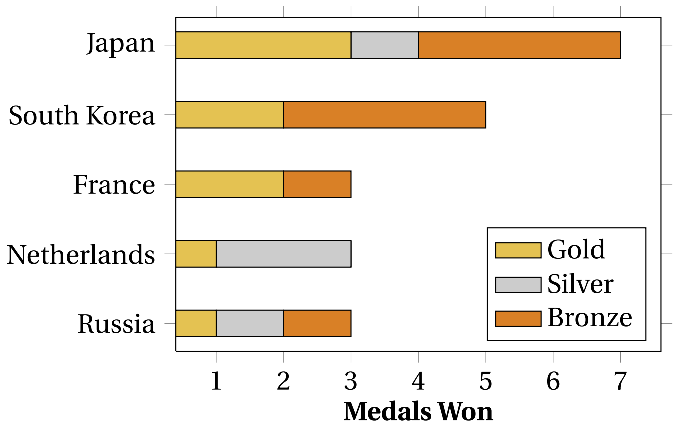
For the medal ranking example, it is easy to compare the contribution of the different medals to the overall medal count of a given country. Likewise, it is easy to compare the number of gold, silver, or bronze medals won by different countries. The reason why this works is that all sizes are small and 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 7.7 presents the input which was used to create the component bar graph depicted in Figure 7.6. 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.

**Figure 7.6**: A component bar graph.




Top five countries of medal ranking of the 2009 World Judo Championships. (Source `wikipedia`.)

**Figure 7.7**: 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}
```

## 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 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.

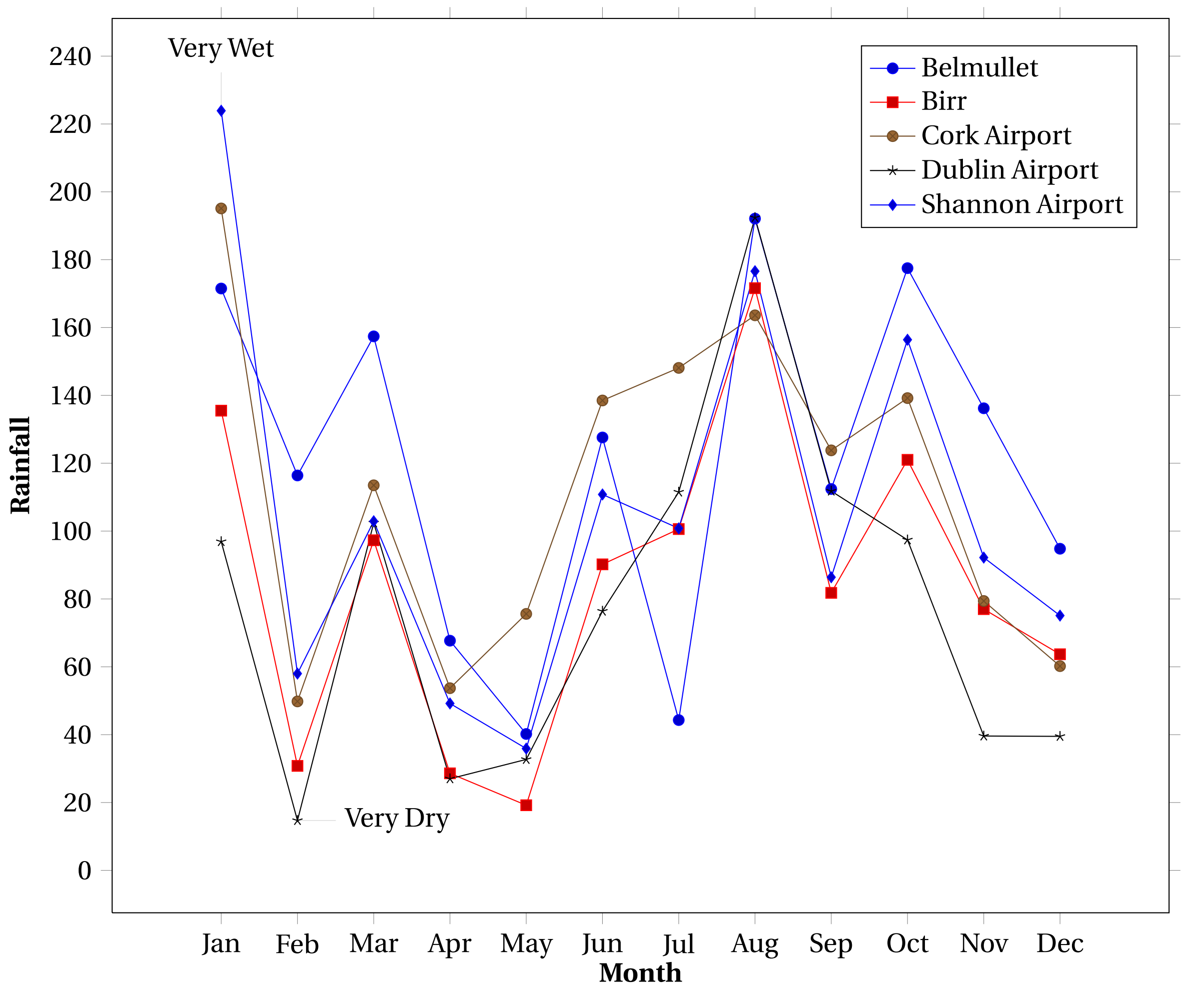
## 7.8 Line Graphs

*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 7.8 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 7.8 direct labelling is virtually impossible.

Figure 7.9 depicts the input which was used to create the line graph in Figure 7.8. 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 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.

**Figure 7.8**: A line graph.




Monthly rainfall in millimetres for the year 2009. (Source http://www.cso.ie.)

**Table 7.1**: Allowed values for `mark` option.

| | | | |
|---|---|---|---|
| | | | **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*` | |

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`.

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 7.10 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 $i$.

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.

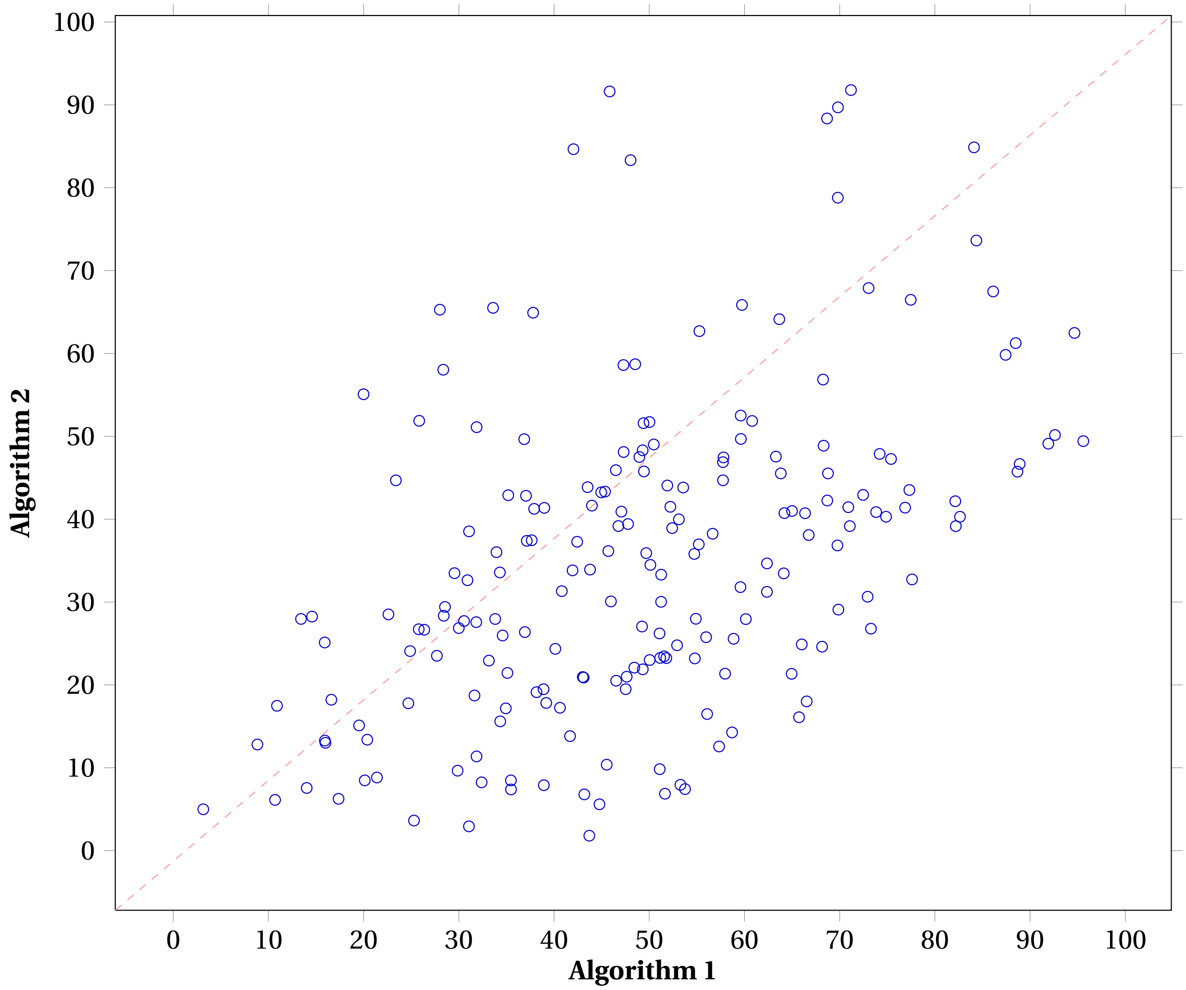
Figure 7.11 presents the code which was used to create the scatter plot in Figure 7.10. The option '`scatter`' states that the coordinates provided by the calls to `\addplot` are for scatter plots. The option '`only marks`' results in a mark which is drawn at each coordinate which is specified by `\addplot`. 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 ex-

**Figure 7.9**: Creating a line graph.

```
\begin{tikzpicture}
\begin{axis}
      [width=\textwidth, enlargelimits=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}
```

**Figure 7.10**: A scatter plot.




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 7.11**: 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}
```

plicit coordinates. Usually scatter plots consist of many data points. Adding all point specifications to the main LaTeX source of your `pgfplot` 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 LaTeX. For further information the reader is referred to a proper LaTeX book such as [Lamport, 1994], a tutorial such as [Oetiker *et al.*, 2007], or a book on using LaTeX for writing mathematics [Voß, 2009]. A comprehensive listing of LaTeX symbols, including math symbols, is provided by [Pakin, 2005].

LaTeX's basic support for mathematics is limited, which is why the AMS are providing 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 $\mathcal{AMS}$-LaTeX software. Section 8.2 describes 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 $\mathcal{AMS}$-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 $\mathcal{AMS}$-LaTeX Platform

$\mathcal{AMS}$-LaTeX 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 LaTeX distribution. You can download the AMS software and documentation from http://www.ams.org/tex/amslatex.html.

The software distributed under the name $\mathcal{AMS}$-LaTeX consists of various extensions for LaTeX. The distribution is divided into two parts:

**amscls:** The acmcls class provides the AMS document class and theorem package. Using this class gives your LaTeX 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 $\mathcal{AMS}$-LaTeX 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 fo 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 $\mathcal{AMS}$-LaTeX are the following. The description of the packages has been adapted from the $\mathcal{AMS}$-LaTeX 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 (\mathfrac).

**amssymb:** Provides lots of extra symbols.

## 8.2 LaTeX's Math Modes

LaTeX 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.

LaTeX Input
```
The Binomial Theorem states that
 $\sum^{n}_{i=0} \binom{n}{i} a^{i} b^{n-i} = (a + b)^{n}$.
Substituting $1$ for $a$ and $1$ for $b$ this gives us
 $\sum^{n}_{i=0} \binom{n}{i} = 2^{n}$.
```

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.

LaTeX Output

The Binomial Theorem states that $\sum^{n}_{i=0} \binom{n}{i} a^{i} b^{n-i} = (a + b)^{n}$. Substituting 1 for $a$ and 1 for $b$ this gives us $\sum^{n}_{i=0} \binom{n}{i} = 2^{n}$.

## 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 (`^`) is for creating superscripts. The expression '`$⟨expr⟩^⟨sup⟩$`' makes `⟨sup⟩` a superscript of `⟨expr⟩`. So '`$e = m c^2$`' gives you '$e = mc^2$'. Grouping works as usual. So to typeset '$e^{a+b}$' you need braces: '`$e^{a+b}$`'.

Subscripts are handled in a similar to superscripts. The subscript operator (`_`) is for creating subscripts. The expression '`$⟨expr⟩_⟨sub⟩$`' makes `⟨sub⟩` a subscript of `⟨expr⟩`. So to get '$f_{n+2} = f_{n+1} + f_n$' you need '`$f_{n + 2} = f_{n + 1} + f_n$`'.

Subscripts and superscripts may be nested and combined. The expressions '`$⟨expr⟩_⟨sub⟩^⟨sup⟩$`' and '`$⟨expr⟩^⟨sup⟩_⟨sub⟩$`' are equivalent and make `⟨sub⟩` a subscript of `⟨expr⟩` and `⟨sup⟩` a superscript of `⟨expr⟩`, so '`$s^{m + 1}_{n+2}$`' gives you '$s^{m+1}_{n+2}$'.

It is good practice to avoid su∗su∗scripts and su∗su∗su∗scripts as much as possible — some style and class files may reject them. The following are some advantages.

**Table 8.1**: Lowercase Greek letters.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | **Standard commands** |
| $\alpha$ | `\alpha` | $\beta$ | `\beta` | $\gamma$ | `\gamma` | $\delta$ | `\delta` | $\epsilon$ | `\epsilon` |
| $\zeta$ | `\zeta` | $\eta$ | `\eta` | $\theta$ | `\theta` | $\iota$ | `\iota` | $\kappa$ | `\kappa` |
| $\lambda$ | `\lambda` | $\mu$ | `\mu` | $\nu$ | `\nu` | $\xi$ | `\xi` | $o$ | `o` |
| $\pi$ | `\pi` | $\rho$ | `\rho` | $\sigma$ | `\sigma` | $\tau$ | `\tau` | $\upsilon$ | `\upsilon` |
| $\phi$ | `\phi` | $\chi$ | `\chi` | $\psi$ | `\psi` | $\omega$ | `\omega` | | |
| | | | | | | | | | `amsmath` **provided commands** |
| $\varepsilon$ | `\varepsilon` | $\vartheta$ | `\vartheta` | $\varkappa$ | `\varkappa` | $\varpi$ | `\varpi` | $\varrho$ | `\varrho` |
| $\varsigma$ | `\varsigma` | $\varphi$ | `\varphi` | | | | | | |
| $\digamma$ | `\digamma` | | | | | | | | |

This table lists the math mode commands for lowercase Greek letters. The commands at the top of the table are standard LaTeX commands. The command `\digamma` and the commands starting with `\var` are provided by AMS-LaTeX.

**Simplicity:** The fewer the su∗scripts, the simpler the notation, the greater the transparency.

**Readability:** The resulting expression is easier to parse.

**Spacing:** Avoiding nested subscripts and superscipts 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 `\alpha` ($\alpha$), `\beta` ($\beta$), `\gamma` ($\gamma$), ....

**Italic:** There are also some commands for italic lowercase Greek letters. These commands all start with `\var`: `\varepsilon` ($\varepsilon$), `\vartheta` ($\vartheta$), `\varrho` ($\varrho$), .... These commands are provided by `amsmath`.

**Dunno:** Finally there is the AMS-LaTeX-provided command `\digamma`, which gives you $\digamma$.

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:

**Regular:** `\Gamma` ($\Gamma$), `\Delta` ($\Delta$), `\Theta` ($\Theta$), .... These commands are standard LaTeX.

**Table 8.2**: Uppercase Greek letters.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | **Standard commands** |
| $\Gamma$ | \Gamma | $\Delta$ | \Delta | $\Theta$ | \Theta | $\Lambda$ | \Lambda | $\Xi$ | \Xi |
| $\Pi$ | \Pi | $\Sigma$ | \Sigma | $\Upsilon$ | \Upsilon | $\Phi$ | \Phi | $\Psi$ | \Psi |
| $\Omega$ | \Omega | | | | | | | | |
| | | | | | | | | | **amsmath provided commands** |
| $\varGamma$ | \varGamma | $\varDelta$ | \varDelta | $\varTheta$ | \varTheta | $\varLambda$ | \varLambda | $\varXi$ | \varXi |
| $\varPi$ | \varPi | $\varSigma$ | \varSigma | $\varUpsilon$ | \varUpsilon | $\varPhi$ | \varPhi | $\varPsi$ | \varPsi |
| $\varOmega$ | \varOmega | | | | | | | | |

This table lists the math mode commands for uppercase Greek letters. The commands at the top of the table are standard LaTeX commands. The commands starting with \var are provided by amsmath.

**Italic:** \varGamma ($\varGamma$), \varDelta ($\varDelta$), \varTheta ($\varTheta$), .... These non-standard commands are provided by AMS-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 LaTeX provides a few commands for displayed math. The 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 referree 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 amsmath'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.

LaTeX 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~$a$ and for~$b$ 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.

LaTeX Output

The following is Newton's Binomial Theorem:

$$\sum^{n}_{i=0}\binom{n}{i}a^{i}b^{n-i}=(a+b)^{n}\,. \tag{8.1}$$

Substituting 1 for $a$ and for $b$ in (8.1) gives us $\sum^{n}_{i=0} \binom{n}{i} = 2^{n}$.

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.

LaTeX Input
```
\begin{equation}
\begin{split}
   a & = b + c + d        \\
     & \qquad + f + g + h \\
     & > 0\,.
\end{split}
\end{equation}
```

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 \qquad, which generates a horizontal space which is roughly equivalent to twice the width of the uppercase letter M. Section 8.21.1 provides more information about how to use the command \qquad.

LaTeX Output

$$\begin{split} a &= b + c + d \\ &\qquad + f + g + h \\ &> 0\,. \end{split} \tag{8.2}$$

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 a 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 more 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

$$\begin{multline} a = b + c + d \\ + f + g + h \\ \shoveleft{+ k + l + m} \\ + n + o + p\,. \tag{8.3}\end{multline}$$

### 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.

LaTeX Input
```
\begin{gather}
   a = b\,,                 \\
   \begin{split}
      a & = m + n + o       \\
        & \qquad + x + y + z \,.
   \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.

LaTeX Output

$$a = b\,, \tag{8.4}$$

$$\begin{split} a & = m + n + o \\ & \qquad + x + y + z\,. \end{split} \tag{8.5}$$

### 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 ($\infty$).

LaTeX Input

```
\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)                          \,.
\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

$$
\begin{aligned}
F &= \sum^\infty_{n=0} f_n z^n && (8.6)\\
&= z + \sum^\infty_{n=2}(f_{n-1}+f_{n-2}) z^n && (8.7)\\
&= z + F/z + F/z^2 && (8.8)\\
&= z / (1 - z - z^2)\,.
\end{aligned}
$$

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\,,  & b_0 & = c_0\,,  & c_0 & = d_0\,,\\
   a_1 & = b_1\,,  & b_1 & = c_1\,,  & c_1 & = d_1\,,\\
   a_2 & = b_2\,,  & b_2 & = c_2\,,  & c_2 & = d_2\,.
\end{align}
```

The following is the resulting output.

LaTeX Output

$$
\begin{aligned}
a_0 &= b_0\,, & b_0 &= c_0\,, & c_0 &= d_0\,, && (8.9)\\
a_1 &= b_1\,, & b_1 &= c_1\,, & c_1 &= d_1\,, && (8.10)\\
a_2 &= b_2\,, & b_2 &= c_2\,, & c_2 &= d_2\,. && (8.11)
\end{aligned}
$$

**Figure 8.1**: The `\intertext` command.

```
\begin{align*}
   x_0 & = 0\,, \\
   x_1 & = 1\,, \\
\intertext{and}
   x_2 & = 2\,.
\end{align*}
```

$$x_0 = 0\,,$$
$$x_1 = 1\,,$$

and

$$x_2 = 2\,.$$

### 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.

### 8.6.7 Interrupting a Display

The `amsmath` package also provides a command called `\intertext` for a short interjection of one or two lines in the middle of a multi-line display. Figure 8.1 demonstrates how it is used. Notice that the equation symbols of the resulting three 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 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 environ-

**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*}
```

$$I = \left[\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}\right]\,.$$

ment it is not always satisfactory. TEXperts 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.

LaTeX Input
```
\[ \text{final}
        = \text{mark for assignments} \times
        + 5 \times \text{mark for literature review} \times\,.  \]
```

This gives you:

LaTeX Output

$$\text{final} = \text{marks for assignments} + 5 \times \text{mark for literature review}\,.$$

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 sections 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 '|' is used as a left and right delimiter in the commonly used notation, $|\cdot|$, for absolute values. Despite the importance of delimiters, 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) LaTeX expression `$f( 2^{2^{2^2}}_{2_{2_2}} )$`. If we typeset it using LaTeX this gives us $f( 2^{2^{2^2}}_{2_{2_2}} )$. The resulting output is not very pretty since the parentheses, which act as delimiters of the arguments of $f(\cdot)$, are too small. LaTeX is simply not aware that the parentheses are serving as delimiters. To tell 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: `$f\left( 2^{2^{2^2}}_{2_{2_2}} \right)$`, which gives us $f\left( 2^{2^{2^2}}_{2_{2_2}} \right)$. 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?

LaTeX Output

$$n! = \left\{ \begin{aligned} & 1 && \text{if $n \leq 1$}\,, \\ & n \times (n-1)! && \text{otherwise}\,. \end{aligned} \right.$$

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.

LaTeX Input

```
\[ n! =
 \left\{
   \begin{aligned}
      & 1               && \text{if $n \leq 1$}\,, \\
      & n \times (n-1)! && \text{otherwise}\,.
   \end{aligned}
 \right. \]
```

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 enviornment 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.

LaTeX Input

```
\begin{align*}
f & = g\left(3^{3^{3}} + ...\right.\\
  & \qquad \left.+ 3 \vphantom{3^{3^{3}} + ...}\right)\,.
\end{align*}
```

If you use this you should get the following:

LaTeX Output

$$f = g\left(3^{3^3} + \ldots\right.$$
$$\left. + 3\right).$$

### 8.8.2 Bars

AMS-LaTeX provides several commands for typesetting vertical bars (|). The reason for having several commands is that LaTeX'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 AMS-LaTeX 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 AMS-LaTeX. The following demonstrates how to typeset the vertical bar which occurs in "guarded" sets.[1]

LaTeX Input

```
The even digits are
 $\{\, 2 i \,\vert\, 0 \leq i \leq 4 \,\}$.
```

The following is the resulting output.

LaTeX Output

The even digits are $\{\, 2 i \,\vert\, 0 \leq i \leq 4 \,\}$.

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, $\{|i|\,|-10 \leq i \leq 9\}$ 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: $\lvert x\rvert$.

**`$\left\lVert x \right\rVert$`**
These commands are for norms: $\lVert x\rVert$.

The `\rvert` command is also used for the "evaluation at" notation.

LaTeX Input

```
\[ \left. f( x ) \right\rvert_{x=0} = 0\,.  \]
```

The following is the resulting output. Notice that the '`\left.`' balances the `\left`-`\right` pair.

LaTeX Output

$$\left. f( x ) \right\rvert_{x=0} = 0\,.$$

[1]The adjective 'guarded' for sets is inspired by guarded lists in the `Haskell` programming language [Peyton Jones and Hughes, 1999].

### 8.8.3 Tuples

A common error in computer science and mathematical papers is to use ‘$<1,2,3>$’ for the tuple/sequence consisting of 1, then 2, and then 3. This kind of LaTeX input gives you ‘$< 1,2,3 >$’, which looks so bad that some authors have complained about this (see for example [Aslaksen, 1993]). LaTeX has a special \langle and \rangle for tuples. If you use them for tuples then the result will look much more aesthetically pleasing. The following provides an example.

LaTeX Input
```
Let $F( z )$ be the ordinary generating function of the sequence
 $\left\langle t_0,t_1,\ldots \right\rangle$, then $z F( z )$
 is the ordinary generating function of the sequence
 $\left\langle 0,t_0,t_1,\ldots \right\rangle$.
```

The following is the resulting output.

LaTeX Output

Let $F(z)$ be the ordinary generating function of the sequence $\langle t_0, t_1, \ldots \rangle$, then $zF(z)$ is the ordinary generating function of the sequence $\langle 0, t_0, t_1, \ldots \rangle$.

### 8.8.4 Floors and Ceilings

The commands \lfloor and \rfloor are for typesetting “floor” expressions which are used to express rounding down. The two related commands \lceil and \rceil are for typesetting “ceiling” expressions. They are for rounding up. The following provides an example.

LaTeX Input
```
Let $x$ be any real. By definition
 $i \leq \left\lfloor x \right\rfloor
    \leq x
    \leq \left\lceil  x \right\rceil
    \leq I$
 for all integers $i$ and $I$ such that $i \leq x \leq I$.
```

The output looks like this.

LaTeX Output

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 \frac. To get $\frac{\langle\text{num}\rangle}{\langle\text{den}\rangle}$ you use \frac{⟨num⟩}{⟨den⟩}. Notice that fractions in the running text may disturb the flow of reading as they may affect the interline spacing. When using the \frac

**Table 8.3**: Variable-size delimiters.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | **Standard** |
| $\{$ | `\{` | $\}$ | `\}` | $\langle$ | `\langle` | $\rangle$ | `\rangle` |
| ( | `(` | ) | `)` | [ | `[` | ] | `]` |
| $\lceil$ | `\lceil` | $\rceil$ | `\rceil` | $\lfloor$ | `\lfloor` | $\rfloor$ | `\rfloor` |
| $\downarrow$ | `\downarrow` | $\Downarrow$ | `\Downarrow` | $\uparrow$ | `\uparrow` | $\Uparrow$ | `\Uparrow` |
| $\updownarrow$ | `\updownarrow` | $\Updownarrow$ | `\Updownarrow` | $\vert$ | `|` | $\Vert$ | `\|` |
| / | `/` | $\backslash$ | `\backslash` | | | | |
| | | | | | | | **amsmath** |
| $\lvert$ | `\lvert` | $\rvert$ | `\rvert` | $\lVert$ | `\lVert` | $\rVert$ | `\rVert` |

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 LaTeX-provided commands. The delimiters listed under the heading '`amsmath`' are provided by AMS-LaTeX. All commands can be used with or without `\left` and `\right`.

command in inline math mode you should ensure that the resulting interline spacing is acceptable. If it is not then perhaps you should turn the `\frac` construct into a simple `$⟨num⟩/⟨den⟩$` 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]{⟨num⟩}{⟨den⟩}` or `\cfrac[r]{⟨num⟩}{⟨den⟩}`. 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.

LaTeX Input
```
\[ F = \cfrac{1}{2 +
       \cfrac{1}{2 +
       \cfrac{1}{2 + \dotsb}}}\,. \]
```

The following is the resulting output.

LaTeX Output

$$F = \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \dotsb}}}\,.$$

## 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 `$\sum^n_{i = 0} f( i )$` 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 $\sum^n_{i=0} f(i)$ is a shorthand for $f(0) + f(1) + f(2) + \cdots + f(n)$.

In the *generalised* summation notation [Graham *et al.*, 1989, Page 22] the range of the index variabe 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

$$\sum_{0\leq k\leq n\,,\mathrm{odd}\ k} 2^k\,.$$

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.

LaTeX Input

```
It is well known that
 $\sum^\infty_{n=0} 2^{-n} = 2$.
However, it is less well known that the Trie Sum, $S_N$,
 satisfies the following property as $N \to \infty$%
 ~\cite[Theorem~4.10]{Sedgewick:Flajolet:96}:
\[ S_N = \sum^\infty_{n = 0}
           \left( 1-\left( 1-2^{-n} \right)^N \right)
       = \lg N + O\left( \log \log N \right)\,.
\]
```

The following is the resulting output. Again notice that the limits are different for ordinary and displayed math.

LaTeX Output

It is well known that $\sum^\infty_{n=0} 2^{-n} = 2$. However, it is less well known that the Trie Sum, $S_N$, satisfies the following property as $N \to \infty$ [Sedgewick and Flajolet, 1996, Theorem 4.10]:

$$S_N = \sum^\infty_{n = 0} \left( 1-\left( 1-2^{-n} \right)^N \right) = \lg N + O\left( \log \log N \right)\,.$$

**Table 8.4**: Variable-sized symbols.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | **standard** |
| $\sum$ | `\sum` | $\int$ | `\int` | $\prod$ | `\prod` | $\coprod$ | `\coprod` |
| $\bigcap$ | `\bigcap` | $\bigcup$ | `\bigcup` | $\bigwedge$ | `\bigwedge` | $\bigvee$ | `\bigvee` |
| $\bigsqcup$ | `\bigsqcup` | $\bigodot$ | `\bigodot` | $\bigotimes$ | `\bigotimes` | $\bigoplus$ | `\bigoplus` |
| $\biguplus$ | `\biguplus` | $\oint$ | `\oint` | | | | |
| | | | | | | | **amsmath** |
| $\iint$ | `\iint` | $\iiint$ | `\iiint` | $\iiiint$ | `\iiiint` | $\idotsint$ | `\idotsint` |

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 LaTeX commands. The commands in the last row of the table are provided by the `amsmath` package.

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 LaTeX 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.

LaTeX Input

```
\[ \frac{\textstyle{\sum^\infty_{n=0}2^{-n}}}{2}
       = 1\,.
\]
```

The following is the resulting output.

LaTeX Output

$$\frac{\textstyle{\sum^\infty_{n=0}2^{-n}}}{2} = 1\,.$$

### 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`.

LaTeX Input
```
\[ \sum_{\substack{\text{$i$ odd}\\0 \leq i\leq n}}
       \binom{n}{i}
 = 2^n -
   \sum_{\substack{\text{$i$ even}\\0\leq i\leq n}}
       \binom{n}{i}\,.
\]
```

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.

LaTeX Output

$$\sum_{\substack{\text{$i$ odd}\\0 \leq i\leq n}} \binom{n}{i} = 2^n - \sum_{\substack{\text{$i$ even}\\0\leq i\leq n}} \binom{n}{i}\,.$$

The `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.

LaTeX Input
```
\[   \sum_{\begin{substack}{l}
                 i \text{␣odd}\\
                 0 \leq i \leq n
           \end{substack}}
         \binom{n}{i}
   = 2^n -
     \sum_{\begin{substack}{c}
                 i \text{␣even}\\
                 0 \leq i \leq n
     \end{substack}}\,. \]
```

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.

LaTeX Output

$$\sum_{\substack{i\text{␣odd}\\0\leq i\leq n}} \binom{n}{i} = 2^n - \sum_{\substack{i\text{␣even}\\0\leq i\leq n}} \binom{n}{i}\,.$$

## 8.11 Functions and Operators

LaTeX comes with a wide range of commands for typesetting functions and operators. However, every TeXnician 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

**Table 8.5**: Log-like functions.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| arccos | `\arccos` | arcsin | `\arcsin` | arctan | `\arctan` | arg | `\arg` |
| cos | `\cos` | cosh | `\cosh` | cot | `\cot` | coth | `\coth` |
| csc | `\csc` | deg | `\deg` | det | `\det` | dim | `\dim` |
| exp | `\exp` | gcd | `\gcd` | hom | `\hom` | inf | `\inf` |
| ker | `\ker` | lg | `\lg` | lim | `\lim` | lim inf | `\liminf` |
| lim sup | `\limsup` | ln | `\ln` | log | `\log` | max | `\max` |
| min | `\min` | Pr | `\Pr` | sec | `\sec` | sin | `\sin` |
| sinh | `\sinh` | sup | `\sup` | tan | `\tan` | tanh | `\tanh` |

**Figure 8.3**: ‘Limit’ argument of log-like functions.

```
\[ \lim_{x \to 0}
        \frac{x^{2}}{x} = 0\,.
\]
```

$$\lim_{x\to 0}\frac{x^{2}}{x} = 0\,.$$

Specifying the ‘limit’ argument of existing log-like functions. The LaTeX input to the left results in the output to the right.

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 $\mathcal{A}\!\mathcal{M}\!\mathcal{S}$-LaTeX-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 LaTeX'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 depending 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 ‘$\gcd(5,3) = \gcd(3,5 \bmod 3)$’.

**\mod**

This is for modular equivalence: '`$2 \equiv 5 \mod 3$`', which gives you '$2 \equiv 5 \mod 3$'. 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.

**\pmod**

This is for parenthesised modular equivalence: '`$2 \equiv 5 \pmod 3$`', which gives you '$2 \equiv 5 \pmod 3$'.

**\pod**

This is for parenthesised modular equivalence without mod symbol: '`$2 \equiv 5 \pod 3$`', which gives you '$2 \equiv 5 \pod 3$'.

### 8.11.2 Declaring New Operators

AMS-LaTeX provides the `\DeclareMathOperator` command for defining new operator names. The command can only be used in the preamble.

LaTeX Usage

```
\DeclareMathOperator{⟨command⟩}{⟨sym⟩}
```

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.

LaTeX Input

```
\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.
```

It will give you the following output.

LaTeX Output

... Note that $1 \mathrm{binop} 2 = 3$ does not look pretty. However, $1 \operatorname{binop} 2 = 3$ looks good.

Notice that the appearance of both operator symbols is the same. However, the spacing for the first operator symbol is dreadful since LaTeX does not recognise it as an operator.

AMS-LaTeX 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.

LaTeX Input

```
\documentclass{article}
\DeclareMathOperator*{\Lim}{Lim}
\begin{document}
 ... $\Lim_{x \to 0} \frac{x^{2}}{x} = 0$. ...
```

It will give you the following output.

LaTeX Output

... $\operatorname{Lim}_{x \to 0} \frac{x^2}{x} = 0$. ...

### 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: $\frac{d}{\mathrm{d}x} f$ versus $\frac{\mathrm{d}f}{\mathrm{d}x}$.
  - How integrals are typeset: $\int f \;\mathrm{d}x$ versus $\int \mathrm{d}x \; f$.
  - 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 thin space before the $\mathrm{d}x$ in the input. (Thin spaces are required for each "d" part, so you write '`\mathop{}\!\mathrm{d} x\mathop{}\!\mathrm{d} y`', and so on.)

LaTeX Input
```
\[ \int^{b}_{a} 3 x^{2}\mathop{}\!\mathrm{d} x
    = \left. x^{3} \right\rvert^{b}_{a}
    = b^{3} - a^{3}\,.
 \]
```

Notice that we could have also written the more terse '`\mathrm dx`' for the '`\mathrm{d} x`'. However, arguably, adding the braces is clearer. The following is the resulting output.

LaTeX Output

$$\int^{b}_{a} 3 x^{2}\mathop{}\!\mathrm{d} x = \left. x^{3} \right\rvert^{b}_{a} = b^{3} - a^{3}\,.$$

The key to typesetting more exotic integrals are the the commands which are provided by the amsmath and the esint packages. Table 8.6 lists these commands.

### 8.12.2 Differentiation

Expressions with differentiations are typeset using the `\frac` command. The expression $\frac{\mathrm{d} u}{\mathrm{d} x}$ may be obtained with `\frac{\mathrm{d} u}{\mathrm{d} x}`. More complex expressions work as expected, so $\frac{\mathrm{d}^{2} u}{\mathrm{d} x^{2}}$ 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.

**Table 8.6**: Integration signs.

| | | | |
|---|---|---|---|
| | | | **amsmath** |
| $\int$ | `\int` | $\iint$ | `\iint` |
| $\iiint$ | `\iiint` | $\iiiint$ | `\iiiint` |
| $\idotsint$ | `\idotsint` | | |
| | | | **esint** |
| $\int$ | `\int` | $\iint$ | `\iint` |
| $\iiint$ | `\iiintop` | $\iiiint$ | `\iiiintop` |
| $\sqint$ | `\sqint` | $\sqiint$ | `\sqiint` |
| $\ointctrclockwise$ | `\ointctrclockwise` | $\ointclockwise$ | `\ointclockwise` |
| $\landupint$ | `\landupint` | $\landdownint$ | `\landdownint` |
| $\fint$ | `\fint` | $\int\cdots\int$ | `\dotsintop` |
| $\oint$ | `\ointop` | $\oiint$ | `\oiintop` |
| $\varointctrclockwise$ | `\varointctrclockwise` | $\varointclockwise$ | `\varointclockwise` |
| $\varoiint$ | `\varoiint` | | |

This table lists integration signs and the commands to typeset them. The first five commands are provided by the `amsmath` package. The remaining commands are provided by the `esint` package.

```
\[ \frac{\partial u}{\partial t}
   = h^{2}
     \left( \frac{\partial^{2} u}{\partial x^{2}}
          + \frac{\partial^{2} u}{\partial y^{2}}
          + \frac{\partial^{2} u}{\partial z^{2}}
     \right)\,.
  \]
```
LaTeX Input

The resulting output is as follows.

LaTeX Output

$$\frac{\partial u}{\partial t} = h^{2} \left( \frac{\partial^{2} u}{\partial x^{2}} + \frac{\partial^{2} u}{\partial y^{2}} + \frac{\partial^{2} u}{\partial z^{2}} \right)\,.$$

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.

LaTeX Output

$\ldots\sqrt{2} \approx 1.414213562$ and $\sqrt[3]{27} = 3$.

Sometimes the placement of the root indices is not perfect: $\sqrt[\beta]{k}$. 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 $\sqrt[\leftroot{-2}\uproot{2}\beta]{k}$.

## 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 LaTeX's built-in `array` environment to typeset a complex-ish construct.

LaTeX Input

```
\[ \left( \begin{array}{c}
     \left\lvert \begin{array}{lrc}
          x & y & z \\
          0 + 1 + 2 & \alpha + \beta + \gamma & a + b + c
     \end{array} \right\rvert \\
          A \\ B
   \end{array}
   \right) \]
```

The resulting output is as follows.

LaTeX Output

$$\left( \begin{array}{c} \left\lvert \begin{array}{lrc} x & y & z \\ 0 + 1 + 2 & \alpha + \beta + \gamma & a + b + c \end{array} \right\rvert \\ A \\ B \end{array} \right)$$

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.

$\mathcal{AMS}$-LaTeX 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.

LaTeX Input
```
... Using matrices the linear transformation
 $\langle\,x,y\,\rangle
  \mapsto
  \langle\,2 x + y, 3 y\,\rangle$
 is written as follows:
 $\bigl[\begin{smallmatrix} 2&1 \\ 0&3 \end{smallmatrix}\bigr]
  \bigl[\begin{smallmatrix} x \\ y \end{smallmatrix}\bigr]$.
 You probably knew this already. ...
```

The following is the corresponding output.

LaTeX Output

... Using matrices the linear transformation $\langle\,x,y\,\rangle \mapsto \langle\,2x+y,3y\,\rangle$ is written as follows: $\bigl[\begin{smallmatrix} 2&1 \\ 0&3 \end{smallmatrix}\bigr]\bigl[\begin{smallmatrix} x \\ y \end{smallmatrix}\bigr]$. You probably 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 ($\overbrace{\langle\texttt{expr}\rangle}$) or underbraces ($\underbrace{\langle\texttt{expr}\rangle}$). The commands for creating such expressions are `\overbrace` and `\underbrace`. As should be clear from the 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.

**Table 8.7**: Math mode accents, hats, and other decorations.

| | | | |
|---|---|---|---|
| | | | **Fixed-size Decorations** |
| $\dot{x}$ | `\dot{x}` | $\acute{x}$ | `\acute{x}` |
| $\ddot{x}$ | `\ddot{x}` | $\grave{x}$ | `\grave{x}` |
| $\dddot{x}$ | `\dddot{x}` | $\hat{x}$ | `\hat{x}` |
| $\ddddot{x}$ | `\ddddot{x}` | $\tilde{x}$ | `\tilde{x}` |
| $\mathring{x}$ | `\mathring{x}` | $\bar{x}$ | `\bar{x}` |
| $\check{x}$ | `\check{x}` | $\vec{x}$ | `\vec{x}` |
| $\breve{x}$ | `\breve{x}` | | |
| | | | **Extensible Decorations** |
| $\overleftarrow{\langle\text{expr}\rangle}$ | `\overleftarrow{⟨expr⟩}` | $\overline{\langle\text{expr}\rangle}$ | `\overline{⟨expr⟩}` |
| $\overrightarrow{\langle\text{expr}\rangle}$ | `\overrightarrow{⟨expr⟩}` | $\widetilde{\langle\text{expr}\rangle}$ | `\widetilde{⟨expr⟩}` |
| $\overleftrightarrow{\langle\text{expr}\rangle}$ | `\overleftrightarrow{⟨expr⟩}` | $\widehat{\langle\text{expr}\rangle}$ | `\widehat{⟨expr⟩}` |
| $\underleftarrow{\langle\text{expr}\rangle}$ | `\underleftarrow{⟨expr⟩}` | $\underbar{\langle\text{expr}\rangle}$ | `\underbar{⟨expr⟩}` |
| $\underleftrightarrow{\langle\text{expr}\rangle}$ | `\underleftrightarrow{⟨expr⟩}` | $\underline{\langle\text{expr}\rangle}$ | `\underline{⟨expr⟩}` |
| $\underrightarrow{\langle\text{expr}\rangle}$ | `\underrightarrow{⟨expr⟩}` | | |

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 `\dddot` and `\ddddot` are provided by `amsmath`.

**\overbrace{⟨under⟩}**

This command gives you $\overbrace{\langle\texttt{under}\rangle}$.

**\overbrace{⟨under⟩}^{⟨over⟩}**

This command results in $\overbrace{\langle\texttt{under}\rangle}^{\langle\texttt{over}\rangle}$.

**\underbrace{⟨under⟩}**

This command gives $\underbrace{\langle\texttt{under}\rangle}$.

**\underbrace{⟨over⟩}_{⟨under⟩}**

This command results in $\underbrace{\langle\texttt{over}\rangle}_{\langle\texttt{under}\rangle}$.

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.)

LaTeX Input
```
\[ \underbrace{1 \times x \times x \times \dotsb \times x}_%
              {\text{$k$~times $\times x$}} = x^{k}\,.
\]
```

The following is the resulting output.

LaTeX Output

$$\underbrace{1 \times x \times x \times \dotsb \times x}_{\text{$k$~times $\times x$}} = x^{k}\,.$$

## 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.

LaTeX Input
```
\[ n! = \begin{cases}
          1                 & \text{if $n = 0$}\,; \\
          (n-1) ! \times n  & \text{if $n > 0$}\,.
        \end{cases}
\]
```

This gives you the following.

LaTeX Output

$$n! = \begin{cases} 1 & \text{if $n = 0$}\,; \\ (n-1)! \times n & \text{if $n > 0$}\,. \end{cases}$$

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 *et al.*, 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.

LaTeX Input
```
... We define
 $n! = [\,n = 0\,] +
  (n-1) ! \times n \times [\,n > 0\,]$. ...
```

This gives you:

LaTeX Output

... 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.

LaTeX Input
```
The successor function,
 $s \colon \mathbb{N} \to \mathbb{N}$,
 is defined as follows:
\[
   s( n ) \mapsto n + 1\,.
\]
```

The following is the resulting output.

LaTeX Output

The successor function, $s \colon \mathbb{N} \to \mathbb{N}$, is defined as follows:

$$s( n ) \mapsto n + 1\,.$$

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 '$s : \mathbb{N} \to \mathbb{N}$'. The `\mathbb` command typesets it 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.

> LaTeX Output
>
> **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 $c$.*

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 ⟨`unit`⟩.⟨`number`⟩, where ⟨`unit`⟩ is the number of the current sectional unit, and ⟨`number`⟩ 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.

> LaTeX Output
>
> **Theorem 1.1** (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 $c$.*

**`definition`:** Usually associated with: Definition, Condition, Problem, and Example. The following demonstrates the appearance of the `definition` style.

> LaTeX Output
>
> **Definition 1.2** (Ceiling). The *ceiling* of real number, $r$, is the smallest integer, $i$, such that $r \leq i$.

**`remark`:** Usually associated with: Remark, Note, Notation, Claim, Summary, Acknowledgement, Case, and Conclusion. The following demonstrates the appearance of the `remark` style.

> LaTeX Output
>
> **Tip 1.3** (Tip). Don't do this at home.

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 ⟨`unit label`⟩.⟨`local`⟩, where ⟨`unit label`⟩ depends on the number/label of the sectional unit (chapter, section, ...), and ⟨`local`⟩ 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.

**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}
```

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 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.

**Table 8.8**: Math mode dot-like symbols.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $.$ | `\ldotp` | $\ldots$ | `\ldots` | $\cdot$ | `\cdotp` | $\cdots$ | `\cdots` |
| $:$ | `\colon` | $\vdots$ | `\vdots` | $\ddots$ | `\ddots` | | |

### 8.19.4 Defining Theorem-like Styles

The command `\newtheoremstyle` is for defining new `amsmath` theorem-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 (□) 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.

LaTeX Input
```
\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}
```

LaTeX Output

*Technical Challenge.* To prove that $3^2 + 4^2 = 5^2$, we note that

$$3^2 + 4^2 = 9 + 4^2 = 9 + 16 = 25 = 5^2\,. \qquad \square$$

## 8.20 Mathematical Punctuation

LaTeX provides several commands for typesetting dot-like symbols. Table 8.8 lists LaTeX'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.

**`\ldotp`**
: Used for the definition of `\ldots` [Knuth, 1990, Page 438].

**`\ldots`**
: Low dots. Used between commas, and when things are juxtaposed with no signs between them at all [Knuth, 1990, Page 172]. For example, `$f(x_{1}, \ldots, x_{n} )$` gives you $f(x_1,\ldots,x_n)$ and `$n(n-1)\ldots(1)$` gives you $n(n-1)\ldots(1)$.

**\cdotp**
Used for the definition of \cdots [Knuth, 1990, Page 438].

**\cdots**
Centred dots. Used between + and − and × signs, between = signs and other binary relational operator signs [Knuth, 1990, Page 172]. For example, `$x_{1}+\cdots+x_{n}$` gives you $x_1 + \cdots + x_n$.

**\colon**
Used as punctuation mark [Knuth, 1990, Page 134]: `$f \colon A \to B$`.

**\ddots**
Used in arrays and matrices.

**\vdots**
Used in arrays and matrices.

Notice that the command \cdot also produces a dot. However, this is not used for punctuation. It is generally used in expressions like $(x_1, \ldots, x_n) \cdot (y_1, \ldots, y_n)$.

Many symbols occurring in mathematical formulae require different spacing depending on their context. The commands which reproduce these symbols are context-unaware. The amsmath package provides several commands to overcome this problem. The following commands are for typesetting dots and sequences of dots.

**\dotsc**
For dots in combination with commas.

**\dotsb**
For dots in combination with binary operators/relations.

**\dotsm**
For multiplication dots.

**\dotsi**
For dots with integrals.

**\dotso:**
For other dots.

The following example is based on the amsmath documentation [American Mathematical Society, 2002].

LaTeX Input
```
... \ldots Then we have series $A_1, A_2, \dotsc$,
   regional sum $A_1 +A_2 +\dotsb$, orthogonal
   product $A_1 A_2 \dotsm$, and infinite integral
   \[ \int_{A_1} \! \int_{A_2} \dotsi\,.\]
```

LaTeX Output

......Then we have series $A_1, A_2, \dotsc$, regional sum $A_1 + A_2 + \dotsb$, orthogonal product $A_1 A_2 \dotsm$, and infinite integral

$$\int_{A_1} \! \int_{A_2} \dotsi\,.$$

## 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

LaTeX 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.

LaTeX Usage
```
for $x = f( a, b )$, $f( b, c )$, or~$f( b, c )$.
```

However, the following is not correct.

Don't Try this at Home
```
for $x = f( a, b ), f( b, c )$, or $f( b, c )$.
```

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 *qquad* after the alignment position. Here a qquad is equivalent to two quads. One quad is the equivalent of the with of the uppercase 'M'. If the continuation line is short you may even consider inserting several qquads. You insert a single quad with the command `\quad`. A single qquad is inserted with the command `\qquad`.

  LaTeX Usage
  ```
  \begin{align*}
   f( x ) & = a + b + c + d \\
          & \qquad + e + f + g\,.
  \end{align*}
  ```

- Linebreaks in products should occur *after* the multiplication operator. The operator should be *repeated* on the next line.

  LaTeX Usage
  ```
  \begin{align*}
   f( x ) & = a \times b \times c \times d \times\\
          & \qquad \times e \times f \times g\,.
  \end{align*}
  ```

### 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.

LaTeX Usage
```
The Fibonacci numbers satisfy
 $F_{n} = F_{n - 1} + F_{n - 2}$,
 \ $n \geq 2$.
```

However, it is probably better to turn the previous example into a proper sentence as follows.

LaTeX Usage
```
The Fibonacci numbers satisfy
 $F_{n} = F_{n - 1} + F_{n - 2}$,
 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.

LaTeX Usage
```
\[ z^{m} G( z )
     = \sum_{n} g_{n - m} z^{n}\,,
       \qquad\text{integer $m \geq 0$}\,.
\]
```

Alternatively, you can put the condition in parentheses. However, if you do this, you have to omit the comma before the condition.

LaTeX Usage
```
\[ z^{m} G( z )
     = \sum_{n} g_{n - m} z^{n}
       \qquad\text{(integer $m \geq 0$)}\,.
\]
```

### 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.

LaTeX Usage
```
$g = 9.8\,\mathrm{m}/\mathrm{s}^{2}$
```

The `siunitx` package provides support for typesetting units. Using the package you write '`\SI{9.8}{\metre\per\second\squared}`'. This gives you $9.8\,\mathrm{m}\,\mathrm{s}^{-2}$ as standard, or $9.8\,\mathrm{m}/\mathrm{s}^{2}$ by setting '`per=slash`' with the `\sisetup` macro. More information about the `siunitx` package may be found in the package documentation [Write, 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.

LaTeX Usage
```
The natural numbers, $\mathbb{N}$, are defined
 $\mathbb{N} = \{ 0, 1, 2, \ldots \}$.
```

**Table 8.9**: Positive spacing commands.

| Command | Output |
|---|---|
| `$\rhd\lhd$` | $\rhd\lhd$ |
| `$\rhd␣\lhd$` | $\rhd \lhd$ |
| `$\rhd\texttt{\textvisiblespace}\lhd$` | $\rhd$␣$\lhd$ |
| `$\rhd\,\lhd$` | $\rhd\,\lhd$ |
| `$\rhd\thinspace\lhd$` | $\rhd\,\lhd$ |
| `$\rhd\:\lhd$` | $\rhd\:\lhd$ |
| `$\rhd\medspace\lhd$` | $\rhd\:\lhd$ |
| `$\rhd\;\lhd$` | $\rhd\;\lhd$ |
| `$\rhd\thickspace\lhd$` | $\rhd\;\lhd$ |
| `$\rhd\quad\lhd$` | $\rhd\quad\lhd$ |
| `$\rhd\qquad\lhd$` | $\rhd\qquad\lhd$ |
| `$\rhd\hphantom{2^{2^{2^2}}}\lhd$` | $\rhd\hphantom{2^{2^{2^2}}}\lhd$ |
| `$\rhd 2^{2^{2^2}}\lhd$` | $\rhd 2^{2^{2^2}}\lhd$ |

This table shows the effect of positive math mode spacing commands. The expressions in the first column are typeset in the right column, which is aligned to the right. The first three rows and the last row in the table are for reference purposes.

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.

LaTeX Usage
```
The even numbers, $E$, are defined
 $E = \left{\, 2 n \,:\, n \in \mathbb{N} \,\right}$.
```

If you don't like the colon then you should write

LaTeX Usage
```
The even numbers, $E$, are defined
 $E = \left{\, 2 n \,\mid\, n \in \mathbb{N} \,\right}$.
```

### 8.21.5 More Spacing Commands

Tables 8.9–8.9 list some horizontal spacing commands which can be used in math mode. The commands in Table 8.9 produce positive spacing. The commands in Table 8.10 produce negative spacing. 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': $\mathit{italic+abc^2}$.

**Table 8.10**: Negative spacing commands.

| Command | Output |
|---|---:|
| `$\rhd\lhd$` | $\rhd\lhd$ |
| `$\rhd␣\lhd$` | $\rhd \lhd$ |
| `$\rhd\!\lhd$` | $\rhd\!\lhd$ |
| `$\rhd\negmedspace\lhd$` | $\rhd\negmedspace\lhd$ |
| `$\rhd\negthickspace\lhd$` | $\rhd\negthickspace\lhd$ |

This table shows the effect of negative math mode spacing commands. The expressions in the first column are typeset in the right column, which is aligned to the right. The first two rows in the table are for reference purposes.

**`$\mathrm{roman + abc^2}$`**
This typesets its argument in 'math roman': $\mathrm{roman + abc^2}$.

**`$\mathbf{bold + abc^2}$`**
This typesets its argument in 'math bold face': $\mathbf{bold + abc^2}$. 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.

**`$\mathsf{sans serif + abc^2}$`**
This typesets its argument in 'math sans serif': $\mathsf{sans serif + abc^2}$.

**`$\mathtt{typewriter + abc^2}$`**
This typesets its argument in 'math tele type': $\mathtt{typewriter + abc^2}$.

**`$\mathcal{CALLIGRAPHIC}$`**
This typesets its argument in 'math calligraphic': $\mathcal{CALLIGRAPHIC}$. The calligraphic letters only come in uppercase.

## 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

LaTeX provides several symbols for binary operations. Table 8.11 lists them all.

### 8.23.2 Relation Symbols

The symbols for relations you get with LaTeX is quite impressive. Table 8.12 lists LaTeX's built-in symbols for binary relations. Additional commands which are provided by `amsmath` are listed in

**Table 8.11**: Binary operation symbols.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\pm$ | `\pm` | $\cap$ | `\cap` | $\diamond$ | `\diamond` | $\oplus$ | `\oplus` |
| $\mp$ | `\mp` | $\cup$ | `\cup` | $\bigtriangleup$ | `\bigtriangleup` | $\ominus$ | `\ominus` |
| $\times$ | `\times` | $\uplus$ | `\uplus` | $\bigtriangledown$ | `\bigtriangledown` | $\otimes$ | `\otimes` |
| $\div$ | `\div` | $\sqcap$ | `\sqcap` | $\triangleleft$ | `\triangleleft` | $\oslash$ | `\oslash` |
| $\ast$ | `\ast` | $\sqcup$ | `\sqcup` | $\triangleright$ | `\triangleright` | $\odot$ | `\odot` |
| $\star$ | `\star` | $\vee$ | `\vee` | $\lhd$ | `\lhd` | $\bigcirc$ | `\bigcirc` |
| $\circ$ | `\circ` | $\wedge$ | `\wedge` | $\rhd$ | `\rhd` | $\dagger$ | `\dagger` |
| $\bullet$ | `\bullet` | $\setminus$ | `\setminus` | $\unlhd$ | `\unlhd` | $\ddagger$ | `\ddagger` |
| $\cdot$ | `\cdot` | $\wr$ | `\wr` | $\unrhd$ | `\unrhd` | $\amalg$ | `\amalg` |

This table lists binary operation symbols and the commands that typeset them. The commands `\lhd`, `\rhd`, `\unlhd`, and `\unlhd` are provided by the `amssymb` package.

**Table 8.12**: Relation symbols.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $<$ | `<` | $=$ | `=` | $>$ | `>` | $\approx$ | `\approx` | $\asymp$ | `\asymp` |
| $\bowtie$ | `\bowtie` | $\cong$ | `\cong` | $\dashv$ | `\dashv` | $\doteq$ | `\doteq` | $\equiv$ | `\equiv` |
| $\frown$ | `\frown` | $\geq$ | `\geq` | $\gg$ | `\gg` | $\in$ | `\in` | $\Join$ | `\Join` |
| $\leq$ | `\leq` | $\ll$ | `\ll` | $\mid$ | `\mid` | $\models$ | `\models` | $\neq$ | `\neq` |
| $\ni$ | `\ni` | $\notin$ | `\notin` | $\parallel$ | `\parallel` | $\perp$ | `\perp` | $\preceq$ | `\preceq` |
| $\prec$ | `\prec` | $\propto$ | `\propto` | $\simeq$ | `\simeq` | $\sim$ | `\sim` | $\smile$ | `\smile` |
| $\sqsubseteq$ | `\sqsubseteq` | $\sqsubset$ | `\sqsubset` | $\sqsupseteq$ | `\sqsupseteq` | $\sqsupset$ | `\sqsupset` | $\subseteq$ | `\subseteq` |
| $\subset$ | `\subset` | $\succeq$ | `\succeq` | $\succ$ | `\succ` | $\supseteq$ | `\supseteq` | $\supset$ | `\supset` |
| $\vdash$ | `\vdash` | | | | | | | | |

This table lists relation symbols and the commands to typeset them. The commands `\Join`, `\sqsubset`, and `\sqsupset` are provided by the `amssymb` package.

Table 8.13.

### 8.23.3 Arrows

LaTeX defines several commands for drawing arrows. All these commands produce fixed-size arrows. Extensible arrows are provided by additional packages. Table 8.14 lists all LaTeX 's built-in commands for fixed-size arrows. Some commands for extensible arrows are listed in Tables 8.15–8.15. These commands, some of which accept an optional argument, require additional packages.

### 8.23.4 Miscellaneous Symbols

Table 8.18 lists LaTeX's "miscellaneous" symbols. It is worthwhile pointing out that the command `\imath` and `\jmath` produce a dotless $\imath$ and a dotless $\jmath$. These symbols should be used in combination with hats and similar decorations. The following example, should show why.

**Table 8.13**: Additional amsmath-provided relation symbols.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\approxeq$ | \approxeq | $\backepsilon$ | \backepsilon | $\backsim$ | \backsim | $\backsimeq$ | \backsimeq |
| $\because$ | \because | $\between$ | \between | $\Bumpeq$ | \Bumpeq | $\bumpeq$ | \bumpeq |
| $\circeq$ | \circeq | $\curlyeqprec$ | \curlyeqprec | $\curlyeqsucc$ | \curlyeqsucc | $\doteqdot$ | \doteqdot |
| $\eqcirc$ | \eqcirc | $\fallingdotseq$ | \fallingdotseq | $\multimap$ | \multimap | $\pitchfork$ | \pitchfork |
| $\precapprox$ | \precapprox | $\preccurlyeq$ | \preccurlyeq | $\precsim$ | \precsim | $\risingdotseq$ | \risingdotseq |
| $\shortmid$ | \shortmid | $\shortparallel$ | \shortparallel | $\smallfrown$ | \smallfrown | $\smallsmile$ | \smallsmile |
| $\succapprox$ | succapprox | $\succcurlyeq$ | \succcurlyeq | $\succsim$ | \succsim | $\therefore$ | \therefore |
| $\thickapprox$ | \thickapprox | $\thicksim$ | \thicksim | $\varpropto$ | \varpropto | $\Vdash$ | \Vdash |
| $\vDash$ | \vDash | $\Vvdash$ | \Vvdash | | | | |

**Table 8.14**: Fixed-size arrow symbols.

| | | | |
|---|---|---|---|
| $\downarrow$ | \downarrow | $\Downarrow$ | \Downarrow |
| $\uparrow$ | \uparrow | $\Uparrow$ | \Uparrow |
| $\updownarrow$ | \updownarrow | $\Updownarrow$ | \Updownarrow |
| $\leftarrow$ | \leftarrow | $\Leftarrow$ | \Leftarrow |
| $\rightarrow$ | \rightarrow | $\Rightarrow$ | \Rightarrow |
| $\longleftarrow$ | \longleftarrow | $\Longleftarrow$ | \Longleftarrow |
| $\longrightarrow$ | \longrightarrow | $\Longrightarrow$ | \Longrightarrow |
| $\leftrightarrow$ | \leftrightarrow | $\Leftrightarrow$ | \Leftrightarrow |
| $\longleftrightarrow$ | \longleftrightarrow | $\Longleftrightarrow$ | \Longleftrightarrow |
| $\mapsto$ | \mapsto | $\hookleftarrow$ | \hookleftarrow |
| $\longmapsto$ | \longmapsto | $\hookrightarrow$ | \hookrightarrow |
| $\leftharpoonup$ | \leftharpoonup | $\nearrow$ | \nearrow |
| $\leftharpoondown$ | \leftharpoondown | $\searrow$ | \searrow |
| $\rightharpoonup$ | \rightharpoonup | $\swarrow$ | \swarrow |
| $\rightharpoondown$ | \rightharpoondown | $\nwarrow$ | \nwarrow |
| $\rightleftharpoons$ | \rightleftharpoons | | |

**Table 8.15**: Non-standard extensible amsmath arrow symbols.

| | | | |
|---|---|---|---|
| $\xleftarrow{\langle\text{expr}\rangle}$ | \xleftarrow{⟨expr⟩} | $\xleftarrow[\langle\text{opt}\rangle]{\langle\text{expr}\rangle}$ | \xleftarrow[⟨opt⟩]{⟨expr⟩} |
| $\xrightarrow{\langle\text{expr}\rangle}$ | \xrightarrow{⟨expr⟩} | $\xrightarrow[\langle\text{opt}\rangle]{\langle\text{expr}\rangle}$ | \xrightarrow[⟨opt⟩]{⟨expr⟩} |
| $\underleftarrow{\langle\text{expr}\rangle}$ | \underleftarrow{⟨expr⟩} | $\underrightarrow{\langle\text{expr}\rangle}$ | \underrightarrow{⟨expr⟩} |
| $\overleftrightarrow{\langle\text{expr}\rangle}$ | \overleftrightarrow{⟨expr⟩} | $\underleftrightarrow{\langle\text{expr}\rangle}$ | \underleftrightarrow{⟨expr⟩} |

**Table 8.16**: Non-standard extensible mathtools arrow symbols.

| | | | |
|---|---|---|---|
| $\xleftrightharpoons{\langle\text{expr}\rangle}$ | \xleftrightharpoons{⟨expr⟩} | $\xrightleftharpoons{\langle\text{expr}\rangle}$ | \xrightleftharpoons{⟨expr⟩} |
| $\xleftharpoondown{\langle\text{expr}\rangle}$ | \xleftharpoondown{⟨expr⟩} | $\xrightharpoondown{\langle\text{expr}\rangle}$ | \xrightharpoondown{⟨expr⟩} |
| $\xleftharpoonup{\langle\text{expr}\rangle}$ | \xleftharpoonup{⟨expr⟩} | $\xrightharpoonup{\langle\text{expr}\rangle}$ | \xrightharpoonup{⟨expr⟩} |
| $\xleftrightarrow{\langle\text{expr}\rangle}$ | \xleftrightarrow{⟨expr⟩} | $\xLeftrightarrow{\langle\text{expr}\rangle}$ | \xLeftrightarrow{⟨expr⟩} |
| $\xhookleftarrow{\langle\text{expr}\rangle}$ | \xhookleftarrow{⟨expr⟩} | $\xhookrightarrow{\langle\text{expr}\rangle}$ | \xhookrightarrow{⟨expr⟩} |
| $\xLeftarrow{\langle\text{expr}\rangle}$ | \xLeftarrow{⟨expr⟩} | $\xRightarrow{\langle\text{expr}\rangle}$ | \xRightarrow{⟨expr⟩} |
| $\xmapsto{\langle\text{expr}\rangle}$ | \xmapsto{⟨expr⟩} | | |

This table lists non-standard mathtools-provided extensible arrow symbols and the commands to typeset them. All these commands also take an optional argument. The versions with options are listed in Table 8.17.

**Table 8.17**: Non-standard extensible mathtools arrow symbols.

| | |
|---|---|
| $\xleftrightharpoons[\langle\text{opt}\rangle]{\langle\text{expr}\rangle}$ | \xleftrightharpoons[⟨opt⟩]{⟨expr⟩} |
| $\xleftrightharpoons[\langle\text{opt}\rangle]{\langle\text{expr}\rangle}$ | \xleftrightharpoons[⟨opt⟩]{⟨expr⟩} |
| $\xrightleftharpoons[\langle\text{opt}\rangle]{\langle\text{expr}\rangle}$ | \xrightleftharpoons[⟨opt⟩]{⟨expr⟩} |
| $\xleftharpoondown[\langle\text{opt}\rangle]{\langle\text{expr}\rangle}$ | \xleftharpoondown[⟨opt⟩]{⟨expr⟩} |
| $\xrightharpoondown[\langle\text{opt}\rangle]{\langle\text{expr}\rangle}$ | \xrightharpoondown[⟨opt⟩]{⟨expr⟩} |
| $\xleftharpoonup[\langle\text{opt}\rangle]{\langle\text{expr}\rangle}$ | \xleftharpoonup[⟨opt⟩]{⟨expr⟩} |
| $\xrightharpoonup[\langle\text{opt}\rangle]{\langle\text{expr}\rangle}$ | \xrightharpoonup[⟨opt⟩]{⟨expr⟩} |
| $\xleftrightarrow[\langle\text{opt}\rangle]{\langle\text{expr}\rangle}$ | \xleftrightarrow[⟨opt⟩]{⟨expr⟩} |
| $\xLeftrightarrow[\langle\text{opt}\rangle]{\langle\text{expr}\rangle}$ | \xLeftrightarrow[⟨opt⟩]{⟨expr⟩} |
| $\xhookleftarrow[\langle\text{opt}\rangle]{\langle\text{expr}\rangle}$ | \xhookleftarrow[⟨opt⟩]{⟨expr⟩} |
| $\xhookrightarrow[\langle\text{opt}\rangle]{\langle\text{expr}\rangle}$ | \xhookrightarrow[⟨opt⟩]{⟨expr⟩} |
| $\xLeftarrow[\langle\text{opt}\rangle]{\langle\text{expr}\rangle}$ | \xLeftarrow[⟨opt⟩]{⟨expr⟩} |
| $\xRightarrow[\langle\text{opt}\rangle]{\langle\text{expr}\rangle}$ | \xRightarrow[⟨opt⟩]{⟨expr⟩} |
| $\xmapsto[\langle\text{opt}\rangle]{\langle\text{expr}\rangle}$ | \xmapsto[⟨opt⟩]{⟨expr⟩} |

This table lists non-standard mathtools-provided extensible arrow symbols and the commands to typeset them. Table 8.16 lists how these commands work without the optional argument.

**Table 8.18**: Miscellaneous symbols.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\exists$ | `\exists` | $\Re$ | `\Re` | $\aleph$ | `\aleph` | $\backslash$ | `\backslash` | $\hbar$ | `\hbar` |
| $\forall$ | `\forall` | $\Im$ | `\Im` | $\wp$ | `\wp` | $\|$ | `\|` | $\imath$ | `\imath` |
| $\neg$ | `\neg` | $\top$ | `\top` | $\Box$ | `\Box` | $\surd$ | `\surd` | $\jmath$ | `\jmath` |
| $\clubsuit$ | `\clubsuit` | $\bot$ | `\bot` | $\Diamond$ | `\Diamond` | $\emptyset$ | `\emptyset` | $\ell$ | `\ell` |
| $\diamondsuit$ | `\diamondsuit` | $\flat$ | `\flat` | $\triangle$ | `\triangle` | $\infty$ | `\infty` | $\prime$ | `\prime` |
| $\heartsuit$ | `\heartsuit` | $\natural$ | `\natural` | $\mho$ | `\mho` | $\angle$ | `\angle` | $\partial$ | `\partial` |
| $\spadesuit$ | `\spadesuit` | $\sharp$ | `\sharp` | $\nabla$ | `\nabla` | | | | |

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.

LaTeX Input

```
Some people write $\hat{i} + \hat{j}$
 but I prefer $\hat{\imath} + \hat{\jmath}$.
```

The following is the output.

LaTeX Output

Some people write $\hat{i} + \hat{j}$ but I prefer $\hat{\imath} + \hat{\jmath}$.

It is interesting to point out that it is easier to distinguish the symbol '$\ell$' (`$\ell$`) from the digit '1' than it is to distinguish the letter '$l$' (`$l$`) from the digit '1'. This makes '`\ell`' an ideal alternative for the letter '$l$'.

CHAPTER

9

# Algorithms

Algorithms are ubiquitous in computer science papers. Knowing how to present your algorithms increases the chances of getting your ideas across.

There are several LaTeX packages for typesetting algorithms and this entire chapter is devoted to presenting some of these packages. If you don't like these packages then you can alway fall back to the `tabbing` environment, which is explained in Section 2.12.5.

## 9.1 The `algorithm2e` Package

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].

### 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 typesets the block without 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 draws a vertical line parallel to the duration of a block. 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 draws a vertical line parallel to the duration of a block. However, this time the end of the block is indicated by a little "bend" in the line. With this option the keyword

**Figure 9.1**: Effect of the options noline, lined, and vlined.

| | | |
|---|---|---|
| **if** ⟨*cond*⟩ **then**<br>⟨stuff⟩<br>**end** | **if** ⟨*cond*⟩ **then**<br>\| ⟨stuff⟩<br>**end** | **if** ⟨*cond*⟩ **then**<br>└ ⟨stuff⟩ |

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.

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 economiccal 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 evironment 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.

**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.

**Figure 9.2**: Using algorithm2e.

```
\begin{algorithm2e}[H]
\KwIn{Integers $a \geq 0$ and $b \geq 0$}
\KwOut{\textsc{Gcd} of $a$ and $b$}
\While{$b \neq 0$}{
    $r \leftarrow a \bmod b$\;
    $a \leftarrow b$\;
    $b \leftarrow r$\;
}
\caption{Euclidean Algorithm}
\end{algorithm2e}
```

**Input**: Integers $a \geq 0$ and $b \geq 0$
**Output**: GCD of $a$ and $b$
**while** $b \neq 0$ **do**
  $r \leftarrow a \bmod b$;
  $a \leftarrow b$;
  $b \leftarrow r$;
**Algorithm 1**: Euclidean Algorithm

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’:

LaTeX Usage
```
\begin{algorithm2e}[tbp]
...
\end{algorithm2e}
```

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 command 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 command typesets the value of the ‘Out’ label followed by ⟨output⟩.

**\KwData{⟨input⟩}**
This command typesets the value of the ‘Data’ label followed by ⟨input⟩.

**\KwResult{⟨output⟩}**
This command typesets the value of the ‘Result’ label followed by ⟨output⟩.

**\KwRet{⟨value⟩}**

This command typesets the value of the 'Ret' label followed by ⟨value⟩. This 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 branching condition 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 as been omited.

LaTeX Output

**if** *⟨condition⟩* **then**
⌊ ⟨clause⟩

**\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.

LaTeX Output

**if** *⟨condition⟩* **then**
| ⟨clause⟩

**\ElseIf(⟨comment⟩){⟨condition⟩}{⟨clause⟩}}**

This typesets a conditional else clause with condition ⟨condition⟩ and final if else clause ⟨clause⟩.

LaTeX Output

**else if** *⟨condition⟩* **then**
⌊ ⟨clause⟩

**\uElseIf(⟨comment⟩){⟨condition⟩}{⟨clause⟩}}**

This typesets a conditional else clause with condition ⟨condition⟩ and non-final else clause ⟨clause⟩.

**Figure 9.3**: Typesetting conditional statements with algorithm2e.

```
\begin{algorithm2e}[tbp]
\uIf{$a < 0$}{
    \tcp{$a < 0$}
} \uElseIf{$a = 0$}{
    \tcp{$a = 0$}
} \lElse\eIf{$a = 1$}{
    \tcp{$a = 1$}
} {
    \tcp{$a > 1$}
}
\end{algorithm2e}
```

**if** $a < 0$ **then**
| // $a < 0$
**else if** $a = 0$ **then**
| // $a = 0$
**else if** $a = 1$ **then**
| // $a = 1$
**else**
| // $a > 1$

LaTeX Output

**else if** ⟨*condition*⟩ **then**
| ⟨clause⟩

**\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.

LaTeX Output

**if** ⟨*condition*⟩ **then**
| ⟨then clause⟩
**else**
| ⟨else clause⟩

**\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.

LaTeX Output

**switch** ⟨*value*⟩ **do**
⟨cases⟩

**\Case**(⟨**comment**⟩){⟨**condition**⟩}{⟨**statements**⟩}

This typesets the final case of the `switch` statement. The following is the resulting output.

LaTeX Output

**case** ⟨*condition*⟩
⟨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.

LaTeX Output

**case** ⟨*condition*⟩
| ⟨statements⟩

**\Other**(⟨**comment**⟩){⟨**statements**⟩}

This typesets the default case of the `switch` statement. The following is the resulting output.

LaTeX Output

**otherwise**
⟨statements⟩

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.

LaTeX Output

**for** ⟨*condition*⟩ **do**
⟨body⟩

**Figure 9.4**: Using algorithm2e's switch statements.

```
\begin{algorithm2e}[tbp]
\Switch{order}{
    \uCase{bloody mary}{
        Add tomato juice\;
        Add wodka\;
        break\;
    }
    \uCase{hot whiskey}{
        Add whiskey\;
        Add hot water\;
        Add lemon and cloves\;
        Add sugar or honey to taste\;
        break\;
    }
    \Other{Serve water\;}
}
\end{algorithm2e}
```

**switch** *order* **do**
- **case** *bloody mary*
  - Add tomato juice;
  - Add wodka;
  - break;
- **case** *hot whiskey*
  - Add whiskey;
  - Add hot water;
  - Add lemon and cloves;
  - Add sugar or honey to taste;
  - break;
- **otherwise**
  - Serve water;

**\ForEach(⟨comment⟩){⟨condition⟩}{⟨body⟩}**

This typesets a `foreach` statement with a "condition" ⟨condition⟩ and body ⟨body⟩. The following is an example of the result.

LaTeX Output

**foreach** ⟨*condition*⟩ **do**
- ⟨body⟩

**\ForAll(⟨comment⟩){⟨condition⟩}{⟨body⟩}**

This typesets a `forall` statement with a "condition" ⟨condition⟩ and body ⟨body⟩. The following is an example of the result.

LaTeX Output

**forall** ⟨*condition*⟩ **do**
- ⟨body⟩

**\While(⟨comment⟩){⟨condition⟩}{⟨body⟩}**

This typesets a `while` statement with condition ⟨condition⟩ and body ⟨body⟩. The following is an example of the result.

LaTeX Output

**while** ⟨*condition*⟩ **do**
- ⟨body⟩

**\Repeat(⟨comment⟩){⟨condition⟩}{⟨body⟩}(⟨comment⟩)**

This typesets a `repeat-until` statement with condition ⟨condition⟩ and body ⟨body⟩. The following is an example of possible output.

LaTeX Output

**repeat**
| ⟨body⟩
**until** *⟨condition⟩* ;

### 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 tpesetting `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.

LaTeX Input

```
\tcp{⟨line one⟩\\
     ⟨line two⟩}
```

LaTeX Output

// ⟨line one⟩
// ⟨line two⟩

**\tcp*{⟨comment⟩}**

This typesets the side comment ⟨comment⟩ right justified. The command `\tcp*[r]{⟨comment⟩}` works analogously.

LaTeX Input

```
⟨statement⟩
\tcp*{⟨comment⟩}
```

LaTeX Output

⟨statement⟩ ;          // ⟨comment⟩

**\tcp*[l]{⟨comment⟩}**

This typesets the side comment ⟨comment⟩ left justified.

LaTeX Input

```
⟨statement⟩
\tcp*[l]{⟨comment⟩}
```

LaTeX Output

⟨statement⟩ ; // ⟨comment⟩

**\tcp*[h]{⟨comment⟩}**

This typesets the comment ⟨comment⟩ left justified in place (here).

LaTeX Input

```
\If(\tcp*[h]{⟨comment⟩})
   {⟨condition⟩}
   {⟨statement⟩}
```

LaTeX Output

**if** *⟨condition⟩* **then** // ⟨comment⟩
⌊ ⟨statement⟩

**\tcp*[f]{⟨comment⟩}**

This typesets the comment ⟨comment⟩ right justified in place (here).

LaTeX Input

```
\If(\tcp*[f]{⟨comment⟩})
   {⟨condition⟩}
   {⟨statement⟩}
```

LaTeX Output

**if** *⟨condition⟩* **then**     // ⟨comment⟩
⌊ ⟨statement⟩

## 9.2 The clrscode Package

This section describes the package clrscode, which typesets algorithms in the same style as is used in [Cormen *et al.*, 2001]. The design of this package is slightly different from algorithm2e. Macros are provided to typeset identifiers that serve different purposes in the algorithms. Indentation is achieved with tabs in a style which is similar to the use of tabs in the tabbing environment. All code is typeset in a single environment which is called codebox.

The remainder of this section provides a short introduction to the package. The content is mainly based on the package documentation [Cormen, 2003].

### 9.2.1 Importing clrscode

The clrscode package has no options so importing it is easy. However, the package does rely on the package latexsym. If you have a good LaTeX installation it is more than likely that this package is already installed.

```
\usepackage{clrscode}
```
LaTeX Input

### 9.2.2 Typesetting Names

### 9.2.3 The codebox Environment

### 9.2.4 Conditional Statements

### 9.2.5 The Switch Statement

### 9.2.6 Iterative Statements

### 9.2.7 Comments

# 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 TEX's four processors and Section 10.4 explains how they process LATEX commands. Section 10.5 explains how to define commands in plain TEX. Section 10.6 presents a common technique for tweaking existing commands. Section 10.7 presents a technique which overcomes the problem that 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

LATEX is a programmable typesetting engine. Commands are the key to controlling your document. The advantages of using commands in LATEX are similar to the advantages of using functions and procedures in high-level programming languages. However, 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. 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 ⟨number$_1$⟩–⟨number$_2$⟩. 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 itemized 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 LaTeX commands, most of which are inherited from TeX.

**Number of arguments:** TeX 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 TeX, 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 TeX 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:** TeX 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

LaTeX 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 TeX parlance ⟨cmd⟩ is called a *command sequence*. A LaTeX 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 LaTeX definitions at definition time. A more detailed description of the expansion of LaTeX 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 LaTeX program which defines a user-defined command `\CTAN` and uses it in the body of the `document` environment.

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 \TeX{} Archive Network`'. Given this definition LaTeX substitutes the substitution text '`Comprehensive \TeX{} Archive Network`' for `\CTAN` each time `\CTAN` is used. The following is the resulting output.

LaTeX Output

I always download my packages from the Comprehensive TeX Archive Network. The Comprehensive 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.

**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}
```

**\newcommand⟨cmd⟩[⟨digit⟩]{⟨subst⟩}**

As before, this defines a new command, ⟨cmd⟩, with substitution text ⟨subst⟩. This time the command takes ⟨digit⟩ 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 ⟨subst⟩. When substituting ⟨subst⟩ for ⟨cmd⟩ TEX's Expansion Processor also substitutes the $i$-th actual argument for $\#i$ in ⟨subst⟩, for $1 \leq i \leq$ ⟨digit⟩. It is not allowed to use $\#i$ in ⟨subst⟩ if i < 1 or ⟨digit⟩ $< i$.

**\renewcommand⟨cmd⟩[⟨digit⟩]{⟨subst⟩}**

This redefines ⟨cmd⟩ as a command with ⟨digit⟩ arguments and substitution text ⟨subst⟩.

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.

**\newcommand⟨cmd⟩[⟨digit⟩][⟨default⟩]{⟨subst⟩}**

This defines a new command sequence, ⟨cmd⟩, with substitution text ⟨subst⟩. As before the command takes ⟨digit⟩ 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 ⟨default⟩.

The command \renewcommand may also be used to define commands with optional arguments: \renewcommand⟨cmd⟩[⟨digit⟩][⟨default⟩]{⟨subst⟩}.

The 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 monospace 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 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 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 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 TeX being written to a file. When this invalid TeX is reread in a subsequent session this may cause errors.

A command is called *robust* if it does not expand to invalid TeX. Otherwise it is called *fragile*.

The command `\protect` is used to prevent 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 TeX Processors

Before studying how LaTeX expands (evaluates) commands, this section briefly revisit the four processors which TeX 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 TEX'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 TEX 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 LATEX applies commands to arguments. Throughout this section it is assumed that the input stream has been tokenised by 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.

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 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{␣ab{c}}`' is given by '`␣ab{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}
```
LaTeX 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 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 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 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 `\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

**Table 10.1**: How expansion works.

| # | Out | In |
|---|---|---|
| 1 | | $\backslash X_1 a_1 b_1 c_1$ |
| 2 | | $\backslash S_1 \{_1 \backslash K_2 \{_2 \backslash S_3 \backslash I_3 \}_2 \}_1 \{_1 \backslash S_2 \{_2 \backslash K_3 \backslash K_3 \}_2 \backslash I_2 \}_1 a_1 b_1 c_1$ |
| 3 | | $\backslash K_1 \{_1 \backslash S_2 \backslash I_2 \}_1 a_1 \{_1 \backslash S_2 \{_2 \backslash K_3 \backslash K_3 \}_2 \backslash I_2 a_2 \}_1 b_1 c_1$ |
| 4 | | $\backslash S_1 \backslash I_1 \{_1 \backslash S_2 \{_2 \backslash K_3 \backslash K_3 \}_2 \backslash I_2 a_2 \}_1 b_1 c_1$ |
| 5 | | $\backslash I_1 b_1 \{_1 \backslash S_2 \{_2 \backslash K_3 \backslash K_3 \}_2 \backslash I_2 a_2 b_2 \}_1 c_1$ |
| 6 | | $\backslash S_1 \backslash K_1 \backslash K_1 b_1 \{_1 \backslash S_2 \{_2 \backslash K_3 \backslash K_3 \}_2 \backslash I_2 a_2 b_2 \}_1 c_1$ |
| 7 | | $\backslash K_1 b_1 \{_1 \backslash K_2 b_2 \}_1 \{_1 \backslash S_2 \{_2 \backslash K_3 \backslash K_3 \}_2 \backslash I_2 a_2 b_2 \}_1 c_1$ |
| 8 | | $b_1 \{_1 \backslash S_2 \{_2 \backslash K_3 \backslash K_3 \}_2 \backslash I_2 a_2 b_2 \}_1 c_1$ |
| 9 | b | $\{_1 \backslash S_2 \{_2 \backslash K_3 \backslash K_3 \}_2 \backslash I_2 a_2 b_2 \}_1 c_1$ |
| 10 | b | $\backslash S_2 \{_2 \backslash K_3 \backslash K_3 \}_2 \backslash I_2 a_2 b_2 \}_1 c_1$ |
| 11 | b | $\backslash K_2 \backslash K_2 a_2 \{_2 \backslash I_3 a_3 \}_2 b_2 \}_1 c_1$ |
| 12 | b | $\backslash K_2 \{_2 \backslash I_3 a_3 \}_2 b_2 \}_1 c_1$ |
| 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 | $\backslash K_2 a_2 \{_2 \backslash K_3 a_3 \}_2 \}_1 c_1$ |
| 16 | b | $a_2 \}_1 c_1$ |
| 17 | ba | $\}_1 c_1$ |
| 18 | ba | $c_1$ |
| | bac | |

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.

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 TEX

In this section we shall study how to define LATEX commands using plain TEX. TEX allows a richer variety of commands than LATEX. The main differences are that 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 TEX command definitions but sometimes they are needed. The best thing to do is define commands using LATEX commands and only define commands with TEX as a final resort.

The following are TEX's commands for defining commands without delimiters.

**`\def`⟨cmd⟩`#1#2...#`*n*`{`⟨subst⟩`}`**

This defines a command, ⟨cmd⟩, with *n* arguments, and with substitution text ⟨subst⟩. 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 TEX command definitions.

**\edef⟨cmd⟩#1#2...#*n*{⟨subst⟩}**

This defines a command, ⟨cmd⟩, with *n* arguments, and substitution text which is the *expansion* of ⟨subst⟩. It should be noted that ⟨subst⟩ is expanded at the time at which ⟨cmd⟩ is defined. The command is local to the group in which it is defined.

The following should explain how the commands `\def` and `\edef` work. Given the definitions in the following listing '`\hello{}␣\ehello`' gives us '`HI hi`'.

LaTeX Usage
```
\def\hi{hi}
\def\hello{\hi}
\edef\ehello{\hi}
\def\hi{HI}
```

Commands which are defined using `\def` or `\edef` are not allowed to have paragraph breaks. To allow paragraph breaks in arguments you add the prefix `\long` to `\def` or `\edef`.

As stated in the explanation of TEX 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 '`\silly01`' gives us '`!0!1!`'.

LaTeX Input
```
\def\silly#1#2{%
    \def\sillier##1{!##1!#2!}\sillier{#1}%
}
```

The following commands are useful when defining low-level commands with TEX.

**\csname␣⟨tokens⟩\endcsname**

This results in the command sequence of the expansion of ⟨tokens⟩. In effect this expands ⟨tokens⟩ and puts a backslash character to the front of the result. For example, `\csname␣command\endcsname` gives `\command`. To see why expansion matters, let's assume we have the definition `\def\ho{Ho}`. With this definition `\csname␣Ho\ho\ho\endcsname` gives us `\HoHoHo`.

**\noexpand⟨token⟩**

This results in ⟨token⟩ without expanding it. For example, the definitions of the commands `\def\hello{\hi}` and `\edef\hello{\noexpand\hi}` are equivalent regardless of the definition of `\hi`.

**\expandafter⟨token⟩⟨tokens⟩**

This expands the first token in ⟨tokens⟩ (using arguments if required) and inserts ⟨token⟩ before the result.

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. The resulting output is 'TEX is excellent and LATEX is brilliant'.

**Figure 10.3**: Defining commands with delimited arguments.

```
\makeatletter % allow @ in commands
\def\cmd#1{%
  \@ifnextchar[%
    {\cmd@relay{#1}}%       use option
    {\cmd@relay{#1}[dflt]}% use default
}
\def\cmd@relay#1[#2]{...}
\makeatother % disallow @ in commands
```

```
\makeatletter
\def\cmd#1{%
  \def\cmd@relay##1[##2]{...}
  \@ifnextchar[%
    {\cmd@relay{#1}}%
    {\cmd@relay{#1}[dflt]}%
}
\makeatother
```

LaTeX Usage

```
\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 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.

LaTeX Usage

```
\def\command|#1|#2|{...}
```

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 delimited argument. In LaTeX terms the example defines a user-defined LaTeX 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, lets assume we have the following definition.

LaTeX Usage

```
\def\agoin{ old chap}
\def\hows#1\agoin{How are you #1?}
```

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 following example 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.

LaTeX Usage

```
\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
```

## 10.7 More than Nine Arguments

As mentioned in Section 10.1, LaTeX does not allow you to have more than nine arguments. This section describes two techniques which helps 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 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.

The following demonstrates the technique.

LaTeX Usage
```
\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
```

The second technique is simpler and implements `\relay` as a local macro. The following demonstrates the technique.

LaTeX Usage
```
\def\cmd#1#2#3#4#5#6#7#8#9{%
  \def\relay##1{Arguments: ##1, ##2, ..., and #1.}%
  \relay%
}
\makeatother
```

## 10.8 Introduction to Environments

This section is about environments. The following are a few reasons in favour of environments.

**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⟩}}. 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'.

The following is 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.

LaTeX Usage

```
\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}
```

CHAPTER 11

# Option Parsing

This short chapter discusses two packages for implementing '⟨key⟩=⟨value⟩' macro interfaces. They overcome several problems with 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 '⟨key⟩=⟨value⟩' interfaces.

## 11.1 Why Use a ⟨Key⟩=⟨Value⟩ Interface?

We've already seen that LaTeX's argument handling mechanism is not ideal. The following are some arguments in favour of ⟨key⟩=⟨value⟩ 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 [Carlisle, 1999b] is one of the more commonly used packages for implementing ⟨key⟩=⟨value⟩ interfaces.

To study the keyval package we shall implement a 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⟩.

LaTeX Usage
```
\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}}
```

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 186 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.

LaTeX Usage
```
\def\compares{%
  \@ifnextchar[%
      {\@compares}%
      {\@compares[]}}
\def\@compares[#1]#2{%
  {\setkeys{compares}{#1}%
   #2\ compares \compares@apples~apples
    with \compares@oranges~oranges.}}
```

Note the extra group within the \@compares command. Its main purpose is keeping the redefinitions 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 ⟨key⟩=⟨value⟩ interfaces. The following are the building blocks.

**\newkeycommand{⟨command⟩}[⟨key-value list⟩][⟨number⟩]{⟨definition⟩}**

This defines a new command ⟨command⟩ that takes ⟨number⟩ *regular* arguments and one optional argument and substitution text ⟨definition⟩. The optional argument is a lis of ⟨key⟩=⟨value⟩ pairs, the keys and default values of which are listed in ⟨key-value list⟩. For each key ⟨key⟩ and default value ⟨default⟩, the argument ⟨key-value list⟩ should have an entry of the form '⟨key⟩=⟨default⟩'. Inside the ⟨definition⟩ you use \commandkey{⟨key⟩} to get the actual value for for the ⟨key⟩. The following implements our command \compares.

LaTeX Usage

```
\newkeycommand{\compares}[apples=3,oranges=2][1]{%
  #1\ compares \commandkey{apples}~apples
    with \commandkey{oranges}~oranges.}
```

**\newkeyenvironment{⟨name⟩}[⟨key-value list⟩][⟨number⟩]{⟨start⟩}{⟨end⟩}**

This defines a new environment ⟨name⟩ with begin and end substitution text ⟨begin⟩ and ⟨end⟩. 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 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 LaTeX. Section 12.5 concludes this chapter by demonstrating how to implement tail-recursion in low-level 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 LaTeX and TeX.

### 12.1.1 Counters

A LaTeX *counter* is a global variable for counting things. The following are the commands related to LaTeX counters.

**`\newcounter{⟨name⟩}`**

This defines a new global *counter*. There a *counter* is a 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 `\include`d [Lamport, 1994, Page 138]. You may only use the command is the document preamble [Lamport, 1994, Page 99], but I've noticed that putting it elsewhere is also allowed.

**`\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.

**\stepcounter{⟨name⟩}**
This increments the counter ⟨name⟩ by one. As with \setcounter, ⟨name⟩ should be the name of an existing counter.

**\addtocounter{⟨name⟩}{⟨increment⟩}**
The 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 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.

LaTeX Usage

```
\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}
```

### 12.1.2 Booleans

LaTeX does not support decision making. To make decisions you need TeX or use a package such as ifthen. In the remainder of this section we shall study TeX's way of decision making. The ifthen package is studied in Section 12.2.

**\newif\if⟨bool⟩**
This is 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 the equivalent of assigning true to the Boolean "variable" ⟨bool⟩.

**Table 12.1**: Length units.

| Unit | Name | Equivalent |
|---|---|---|
| pt | point | |
| pc | pica | 1 pc = 12 pt |
| in | inch | 1 in = 72.27 pt |
| bp | big point | 72 bp = 1 in |
| cm | centimetre | 2.54 cm = 1 in |
| mm | millimetre | 10 mm = 1 cm |
| dd | didôt point | 1157 dd = 1238 pt |
| cc | cicero | 1 cc = 12 dd |
| sp | scaled point | 65536 sp = 1 pt |

**\⟨bool⟩false**

This is the equivalent of assigning `false` to the Boolean "variable" ⟨bool⟩.

**\if⟨bool⟩⟨then clause⟩\fi**

This is 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.

LATEX Usage

```
\newif\ifnotes
\notestrue

\begin{document}
\section{\ifnotes Lecture Notes%
         \else Presentation%
         \fi}
...
\end{document}
```

### 12.1.3 Lengths

This chapter studies *length* variables, which are LATEX/TEX 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].

LATEX 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 ⟨unit⟩ by any constant ⟨constant⟩ by writing '⟨constant⟩⟨unit⟩'. For example, '101in' is equivalent to '256.54cm'.

*Length variables* hold length values. They are denoted as command sequences. Given length variable ⟨len⟩, 2⟨len⟩ 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\rubber` gives you a rigid length which is the equivalent of the length value of `\rubber`.

The following are some of 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.

**`\parindent`**
: This length variable stores the amount of indentation at the beginning of a normal paragraph.

**`\textwidth`**
: This length variable stores the width of the text on the page.

**`\textheight`**
: This length variable stores the height of the body of a page, excluding the head and foot space.

**`\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 cause by `\parskip` usually does not result in additional inter-paragraph spacing. However, it does allow the length to stretch if the `\flushbottom` declaration is in effect.

**`\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 LaTeX's length-related commands.

**`\newlength{⟨command⟩}`**
: This defines the length command ⟨command⟩ with an initial value of `0cm`. For example, the command `\newlength{\mylen}` defines a new length called `\mylen`.

**`\setlength{⟨command⟩}{⟨length⟩}`**
: This assigns the length value ⟨length⟩ to the length command ⟨command⟩. For example, the command `\setlength{\parskip}{1.0mm}` assigns the value `1mm` to `\parskip`.

**`\addtolength{⟨command⟩}{⟨length⟩}`**
: This adds the length value ⟨length⟩ to the current value of the length ⟨command⟩. For example, the spell `\addtolength{\parskip}{1.0mm}` adds a millimetre to `\parskip`.

**\settowidth{⟨command⟩}{⟨stuff⟩}**
This assigns the width of ⟨stuff⟩ to ⟨command⟩. 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 ⟨stuff⟩ to ⟨command⟩. 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 ⟨stuff⟩ to ⟨command⟩. 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, TEX Booleans, and lengths. Counters are *global* which is to say that the values of counter variables are *not* restored upon leaving a group. TEX 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 LATEX 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 ⟨bool⟩ is already defined.

**\provideboolean{⟨bool⟩}**
This also defines a new global Boolean variable. However, this command will accept ⟨bool⟩ if it is already defined.

**\setboolean{⟨bool⟩}{⟨value⟩}**
This assigns the value ⟨value⟩ to ⟨bool⟩. Here ⟨value⟩ 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 ⟨then clause⟩ if ⟨test⟩ evaluates to `true` and carries out ⟨else clause⟩ if ⟨test⟩ evaluates to `false`. The condition ⟨test⟩ must be of the following form:

**⟨boolean⟩**
Here ⟨boolean⟩ should be '`true`' or '`false`', ignoring case, so '`true`', '`truE`', ..., '`TRUe`', and '`TRUE`' are equivalent, and so are '`false`', '`falsE`', ..., '`FALSe`', and '`FALSE`'.

**⟨number$_1$⟩⟨op⟩⟨number$_2$⟩**
Here ⟨number$_1$⟩ and ⟨number$_2$⟩ should be numbers and ⟨op⟩ should be '<', '=', or '>'.

**\lengthtest{⟨dimen$_1$⟩⟨op⟩⟨dimen$_2$⟩}**
Here ⟨dimen$_1$⟩ and ⟨dimen$_2$⟩ should be dimension values and ⟨op⟩ should be '<', '=', or '>'.

**\isodd{⟨number⟩}**
As suggested by the notation ⟨number⟩ should be a number.

**\isundefined{⟨command⟩}**
Here ⟨command⟩ should be a command sequence name.

**\equal{⟨string$_1$⟩}{⟨string$_2$⟩}**
Here ⟨string$_1$⟩ and ⟨string$_2$⟩ are evaluated and compared for equality. The test is equivalent to `true` if and only if the results of the evaluations are equal.

**\boolean{⟨bool⟩}**
Here ⟨bool⟩ should be a Boolean variable.

**⟨test$_1$⟩⟨command⟩⟨test$_1$⟩**
Here ⟨test$_1$⟩ and ⟨test$_2$⟩ should be valid ⟨test⟩ conditions and ⟨command⟩ is `\or`, `\and`, `\OR`, or `\AND`. The versions `\OR` and `\AND` are preferred to `\or` and `\and` as they are more robust.

**⟨negation⟩⟨test⟩**
Here ⟨test⟩ should be a valid ⟨test⟩ condition and ⟨negation⟩ should be '`\not`' or '`\NOT`'. The upper case version is preferred to the lower case version.

**\(⟨test⟩\)**
Here ⟨test⟩ should be a valid ⟨test⟩ condition.

The following example demonstrates how to use the `\ifthenelse` command. The `page` counter variable, which is used in the example, keeps track of LaTeX's page numbers.

LaTeX Usage

```
\usepackage{ifthen}

\begin{document}
\ifthenelse{\isodd{\value{page}}}%
           {We're on an odd page.}%
           {The page is even.}
\end{document}
```

The command `\whiledo{⟨test⟩}{⟨statement⟩}` is `ifthen`'s equivalent of the `while` statement. It repeatedly 'executes' ⟨statement⟩ while ⟨test⟩ evaluates to `true`. The following example demonstrates some of the functionality of the `ifthen` package.

LaTeX Usage

```
\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 TeX and 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 LaTeX equivalent. The interested reader is referred to the package's excellent documentation [Krab Thorub *et al.*, 2005].

## 12.4 Looping

The 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. (Figure 10.3 explains how to enable this.)

**`\@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. Given the definition '`\def\swop#1#2{#2#1}`', the command '`\@tfor\var:=1\swop\do{\var23}`' gives '12332'.

## 12.5 Tail Recursion

This section studies tail recursion and demonstrates how it may be implemented using low-level TeX delimited commands. By carefully studying this section the interested reader should fully appreciate TeX and LaTeX programming in the large. The evaluation of the program which is depicted

**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 aabcbc.
\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}
```

in Figure 12.1, demonstrates TEX 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 Style and Class Files

## 13.1 User-defined Style Files

## 13.2 User-defined Class Files

# Part VI

# Writing, Reviewing, and Presenting

CHAPTER 14

# The Academic Publishing Process

**TO DO:** Get started with this.

## 14.1 The Academic Publishing Process

### 14.1.1 Your Audience

Selecting the proper conference or journal.

### 14.1.2 Writing the Paper

Writing the paper.

### 14.1.3 Submitting the Paper

Submitting the paper.

### 14.1.4 Reviewing the Paper

Reviewing a paper.

### 14.1.5 Publishing the Paper

Publishing a paper.

## 14.2 Criteria for Evaluating a Paper

Criteria for evaluating a paper.

## 14.3 Dealing with Rejections

How to deal with rejections.

## 14.4 Citations

### 14.4.1 Theory of Citations

How to cite other peoples' work. The importance of citations.

### 14.4.2 Common Bibliography Styles

## 14.5 Literature Research

### 14.5.1 How to Conduct a Literature Research

How to conduct your literature research.

### 14.5.2 On-line Bibliography Resources

On-line bibliography resources.

## 14.6 Organising your Text

Organising your text.

## 14.7 Planning and Control

Planning the writing of your text. Setting deadlines.

### 14.7.1 Setting Deadlines

### 14.7.2 Version Control

## 14.8 Writing with Co-authors

### 14.8.1 Advantages

### 14.8.2 Complications

### 14.8.3 Tools

Version control. Agreeing on macros to guarantee consistent notation.

## 14.9 English as a Foreign Language

What if English is not your mother tongue.

## 14.10 Caveats and Tips

Writing tips.

## 14.11 Writing a Thesis

Writing a thesis.

### 14.11.1 More

How to deal with your supervisor?

### 14.11.2 How to get the Best out of your Supervisor

How to make use of and deal with your supervisor?

## 14.12 Ethics

## 14.13 Resources

books about typography, books about spelling and grammar, ....

CHAPTER 15

# Writing Skills

## 15.1 Organising an Article with the `llncs` Class

How to organize an article using the `llncs` class,

## 15.2 Organising Proceedings with the `llncs` Class

How to organize workshop and conference proceedings using the `llncs` class,

## 15.3 Creating a Poster with the `a0` Package

How to create a poster.

CHAPTER 16

# Reviewing Skills

**TO DO:** Get started with this.

Chapter

17

# Presentation Skills

## **TO DO:** Get started with this.

## 17.1 Criteria for Evaluating a Presentation

## 17.2 Take-home Message

## 17.3 Planning the Presentation

## 17.4 Avoiding Stress

## 17.5 Giving the Presentation

### 17.5.1 Introduction

### 17.5.2 Related Work

### 17.5.3 Main Results

### 17.5.4 Conclusion

### 17.5.5 Questions from the Audience

# Part VII

# Miscellany

CHAPTER

18

# Creating Presentations with `beamer`

**TO DO:** Introduction to `beamer`.

Chapter 19

# Installing LaTeX and Friends

This chapter describes how to install a widely available LaTeX distribution called TeX Live, how to configure TeX Live, how to install LaTeX class and style files, and how to install and use new fonts. The remainder of this chapter is as follows. Section 19.1 explains how to install the TeX Live distribution. This is followed by Section 19.2 which explains how to configure the TeX Live distribution. Section 19.3 explains how to install new classes and packages (style files). Section 19.4 explains how to install LaTeX fonts — this is not an easy task. Section 19.5 describes the easier task of installing `Unix` `.otf` and `.ttf` fonts. In Section 19.6 it is shown how these `Unix` fonts can be used directly in LaTeX using the `fontspec` package. Section 19.7. concludes this chapter by providing some clues on how to intall TeX Live with a package manager such as `apt-get`.

## 19.1 Installing TeX Live

One of the easier LaTeX distribution to install is TeX Live. TeX Live may be downloaded from the *Comprehensive TeX Archive Network* (CTAN) from `ftp.heanet.ie/pub/CTAN/tex/systems/texlive/Images/`.

The TeX 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.

`Unix` Session

```
# mkdir /mnt/texlive
# mount -t iso9660 -o loop texlive.iso /mnt/texlive
```

The image being mounted, we continue by going to the mounted directory and by running the install program, which is called `install-tl.sh`:

```
Unix Session
# cd /mnt/texlive
# ./install-tl.sh
```

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.

```
Unix Session
# cd /
# umount /mnt/texlive
# rmdir /mnt/texlive
```

## 19.2 Configuring TEX Live

Having installed the TEX 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 LATEX and friends. Throughout LATEX is used as a shorthand for LATEX and friends. In the following, Section 19.2.1 explains how to configure the `Unix` search path and Section 19.2.2 explains how to configure the LATEX search path.

### 19.2.1 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 LATEX 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`.

```
Unix Session
~> PATH=/usr/local/texlive/2007/bin/i386-linux:${PATH}
~> export PATH
```

The best thing is putting these commands in your `.login` file.

### 19.2.2 Configuring `TEXINPUTS`

The final task of our configuration chores is setting up the LATEX 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 LATEX 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 LATEX will assume that `path` is the directory from which it should load `file`. Otherwise LATEX will try and locate `file` in its default directories. This mechanism allows you to help LATEX locating 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, LaTeX will search the path recursively. The following is a typical configuration, which tells LaTeX to first search the current directory, next recursively search the directory ${HOME}/LaTeX/styles, and finally recursively search the directory ${HOME}/LaTeX/mpost.

Unix Session
```
˜> export LaTeX=${HOME}/LaTeX
˜> export TEXINPUTS=.:${LaTeX}/styles//:${LaTeX}/mpost//:
```

## 19.3 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 19.2.2 — the user can force LaTeX 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 LaTeX 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 LaTeX on the .ins file.

**Create documentation:** Run LaTeX 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 1.7.4 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 LaTeX to find your files on subsequent runs.

## 19.4 Installing LaTeX Fonts

This section briefly explains how to install new fonts. To Do.

## 19.5 Installing Unix Fonts

If you haven't done it before then installing fonts the LaTeX 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.[1]

[1] Some of the features of the fontspec package are described in Section 19.6.

**Figure 19.1**: 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}
```

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 mono-space 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 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.

Unix Session
```
~> su
Password:
# fc-cache -fvr ./fonts
```

## 19.6 Using the fontspec Package

The fontspec package provides an easy mechanism for configuring fonts. It significantly reduces the task of installing fonts. Currently the package can only be used in combination with the xelatex LaTeX clone, which comes with a standard TeX Live distribution. It can also be downloaded from CTAN.

It is impossible to explain the functionality of the fontspec package in full detail. Figure 19.1 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.

## 19.7 Package Managers

With the advance of package managers, such as apt-get (Debian, Ubuntu, ...) installing LaTeX 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 TeX Live.

Installing TeX Live with apt-get requires the following two commands and the typing of the root password:

Unix Session
```
~> sudo apt-get update
[sudo] password for user:
~> sudo apt-get install texlive
```

A full version of TeX Live may be installed as follows:

Unix Session
```
~> sudo apt-get install texlive-full
```

An additional advantage of installing LaTeX with a package manger 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 20

# Resources

This section provides some pointers to useful resources. Most of them are available online.

## 20.1 Books about TEX and LATEX

The following are great books about TEX and LATEX. Many of them are freely available.

- [Knuth, 1990]: The original reference to TEX. The source code is freely available.
- [Lamport, 1994]: From the creator of LATEX.
- [Abrahams *et al.*, 2003]: Good book about TEX. Freely available.
- [Eijkhout, 2007]: Good book about TEX. Freely available.
- [Goossens *et al.*, 1994]: LATEX book.
- [Goossens *et al.*, 1997]: Graphics and LATEX.
- [Goossens *et al.*, 1999]: Creating navigable documents with TEX and LATEX.

## 20.2 Articles by the LATEX3 Team

Articles by the LATEX3 Team. All are available online.

- [Team, 2001b]: LATEX User guide.
- [Team, 2001c]: Explains how to to create LATEX class and style files. A seemingly very related `etoolbox` package.
- [Team, 2001a]: Explains the LATEX Font Selection Mechanism.

- [Team, 2001d]: Explains how to configure LaTeX.
- [Braams *et al.*, 2001]: Describes the LaTeX $2_\varepsilon$ sources.

## 20.3 LaTeX Articles, Course Notes and Tutorials

LaTeX and TeX articles, course notes, and tutorials: All are available online.

- [Oetiker *et al.*, 2007]: Most-cited LaTeX tutorial.
- [Voß, 2009]: Comprehensive and interesting presentation of commands and environments in math mode (including $\mathcal{AMS}$-LaTeX-provided symbols).
- [Eijkhout, 2004]: Course about the computer science behind LaTeX.
- [Krishnan, 2003]: A very comprehensive LaTeX tutorial.
- [Heck, 2005a]: Hands-on LaTeX tutorial.
- [Reckdahl, 2006]: Explains how to include imported graphics in LaTeX and pdfLaTeX.
- [Pakin, 2005]: A list of symbols and how to produce them with LaTeX.
- [Pakin, 2006]: A Frequently Asked Questions (FAQ). Click on the sensitive areas and you'll be shown how to produce it.
- [Maltby, 1992]: A TeX tutorial.

## 20.4 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.
- [Hagen, ]: Describes how to turn LaTeX into METAPOST graphics.
- [Hurlin, 2007]: Practical introduction to METAPOST.
- [Heck, 2005b]: Hands-on introduction to METAPOST.

## 20.5 Writing Resources

- [Ramsey, 2006a; Ramsey, 2006b]: : Student and teacher version of what seems to be a very interesting group-based course on writing. I very much enjoyed reading them.
- [Knuth *et al.*, 1987]: Text on mathematical writing based on a course taught at Stanford.
- [Derntl, 2003]: Seminar for PhD students on writing and publishing.
- [Dupré, 1998]: How to get a feeling of good and bad use of English.
- [Levin and Redell, 1983]:

- [Lee, ]:
- [Conrad, xxxx]: A two-page guide on common errors in mathematical writing. There is also a two page adaptation: [Zorn, ].
- [Aslaksen, 1993]: Ten tips for mathematicians using LaTeX.
- [Peyton Jones, 2004]:
- [Goldreich, 2004]:
- [Goldreich, 1991]:
- [Gopen and Swan, ]:
- [Pugh:1993, 1993]:
- [Kurose, 2006]:
- [van Leunen and Lipton, ]:
- [Halmos, WHEN]:
- [Aceto, 2003]:
- [Walsh, ]:
- On-line Merriam Webster English Dictionary: `http://www.m-w.com/netdict.htm`.

## 20.6 Bibliography Resources

Detailed information about managing your bibliography with LaTeX is [Fenn, 2006]. More information may be found at `http://en.wikipedia.org/wiki/BibTeX`.

## 20.7 On-line Resources

- Comprehensive TeX Archive Network (CTAN):
  - Home: `http://www.ctan.org`.
  - Search: `http://www.ucc.ie/cgi-bin/ctan`.
- TeX User Group (TUG):
  - Home: `http://www.tug.org/`.
  - Resources: `http://www.tug.org/interest.html`.
- UK TeX FAQ: `http://www.tex.ac.uk/cgi-bin/texfaq2html`.
- The American Mathematical Society's TeX 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 TEX): http://www-cs-faculty.stanford.edu/~knuth/.
- John Hobby's METAPOST Pages: http://cm.bell-labs.com/who/hobby/MetaPost.html.
- The LATEX Project: http://www.latex-project.org/.
- PdfTEX support: http://www.tug.org/applications/pdftex/.
- Generating PDF with animations and LATEX: http://darkwing.uoregon.edu/~noeckel/PDFmovie.html.
- A list of METAPOST links may be found at http://csweb.ucc.ie/~dongen/mpost/links.html.
- The ghostview resource page: http://pages.cs.wisc.edu/~ghost/gv/index.htm.
- The gsview page: http://pages.cs.wisc.edu/~ghost/gsview/.
- The AucTEX pages. AucTEX is a 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 TEXnicCenter pages. TEXnicCenter is an integrated LATEX development environment (IDE). The package may be downloaded from http://www.texniccenter.org/.
- On-line Oxford English Dictionary: http://dictionary.oed.com/.

# Part VIII

# References and Bibliography

# Indices

# Index of LaTeX and TeX Commands

## Index of Environments

## Index of Classes

# Index of Packages

## Index of External Commands

# Acronyms

| | |
|---|---|
| **AMS** | American Mathematical Society |
| **APL** | A Programming Language |
| **CTAN** | Comprehensive TeX Archive Network |
| **CD** | Compact Disk |
| **FAQ** | Frequently Asked Questions |
| **GUI** | Graphical User Interfaces |
| **IDE** | Integrated Development Environment |
| **SI** | International System of Units |
| **WYSIWYG** | What You See is What You Get |

# Bibliography


[Abrahams *et al.*, 2003] P. A. Abrahams, K. A. Hargreaves, and K. Berry. *TEX for the Impatient*. Addison-Wesley, 2003. This book is freely available from ftp://tug.org/tex/impatient.

[Aceto, 2003] Luca Aceto. How to write a paper, 2003. This presentation is availble from http://www.cs.auc.dk/~luca/PDK/pdk.html.

[American Mathematical Society, 2002] American Mathematical Society. User's guide for the amsmath package, 2002. Version 2.0.

[American Mathematical Society, 2004] American Mathematical Society. Using the amsthm package, 2004. Version 2.20.

[Aslaksen, 1993] Helmer Aslaksen. Ten TEX tricks for the mathematician. *TUGboat, Communications of the TEX Users Group*, 14:135–136, 1993. A modern version of this paper is available from http://www.math.nus.edu.sg/aslaksen/cs/tug-update.pdf.

[Bigwood and Spore, 2003] Sally Bigwood and Melissa Spore. *Presenting Numbers, Tables, and Charts*. Oxford University Press, 2003.

[Braams *et al.*, 2001] Johannes Braams, David Carlisle, Alan Jeffrey, Leslie Lamport, Frank Mittelbach, Chris Rowley, and Rainer Schöpf. The LATEX 2$_\varepsilon$ sources, 2001. Available from http://www.tug.org/texlive/Contents/live/texmf-dist/doc/latex/base/source2e.pdf.

[Breitenbucher, 2005] Jon Breitenbucher. LATEX at a liberal arts college. *The PracTEX Journal*, 3, 2005. Available from http://www.tug.org/pracjourn/index.html.

[Buchsbaum and Reinaldo, 2007] Arthur Buchsbaum and Francisco Reinaldo. A tool for logicians. *The PracTEX Journal*, 3, 2007. Available from http://www.tug.org/pracjourn/index.html.

[Burt, 2005] John Burt. Using poemscol for critical editions of poetry. *The PracTEX Journal*, 3, 2005. Available from http://www.tug.org/pracjourn/index.html.

[Carlisle and Ratz, 1999] D. P. Carlisle and S. P. Q. Ratz. The `graphicx` package, 1999.

[Carlisle, 1999a] David Carlisle. The `enumerate` package, 1999.

[Carlisle, 1999b] David Carlisle. The `keyval` package, 1999.

[Carlisle, 2003] D. P. Carlisle. Packages in the `graphics` bundle, 2003.

[Chervet, 2009] Florent Chervet. The `keycommand` package, 2009.

[Conrad, xxxx] Keith Conrad. Errors in mathematical writing, xxxx. Available from http://www.math.uconn.edu/~kconrad/math216/mathwriting.pdf.

[Cormen *et al.*, 2001] Thomas H. Cormen, Charles E. Leiserson, Ronal L. Rivest, and Clifford Stein. *Introduction to the Analysis of Algorithms*. MIT Press and McGraw-Hill, 2001. Check Year and add isbn.

[Cormen, 2003] Thomas H. Cormen. Using the `clrscode` package in LaTeX $2_{\varepsilon}$, 2003.

[Dearborn, 2006] Elizabeth Dearborn. TeX for medicine. *The PracTeX Journal*, 4, 2006. Available from http://www.tug.org/pracjourn/index.html.

[Derntl, 2003] Michael Derntl. Basics of research paper writing and publishing, 2003. Available from http://www.pri.univie.ac.at/~derntl/papers/meth-se.pdf.

[Dupré, 1998] L. Dupré. *Bugs in Writing, a Guide to Debugging your Prose*. Addison-Wesley Professional, 1998.

[Eijkhout, 2004] V. Eijkhout. The computer science of TeX and LaTeX, 2004. Computer Science course 594, University of Tennessee, Available from www.cs.utk.edu/~eijkhout/594-LaTeX/handouts/TeX%20LaTeX%20course.pdf.

[Eijkhout, 2007] V. Eijkhout. *TeX by Topic, A TeXnician's Reference*. Addison-Wesley, 2007. This book is freely available from http://www.eijkhout.net/tbt/.

[Fenn, 2006] Jürgen Fenn. Managing citations and your bibliography with BibTeX. *PracTeX Journal*, 2006. Available from http://www.tug.org/pracjourn/2006-4/fenn/.

[Feuersänger, 2008] Christian Feuersänger. Manual for package PGFPLOTSTABLE, 2008. Version 1.1.

[Feuersänger, 2010] Christian Feuersänger. Manual for package PGFPLOTS, 2010. Version 1.3.

[Fine, 1992] Jonathan Fine. Some basic control macros for TeX. *TUGboat*, 13(1), 1992.

[Firio, 2004] Christophe Firio. `algorithm2e.sty` — *release 3.3*, 2004.

[Garcia and Buchsbaum, 2010] Aracele Garcia and Arthur Buchsbaum. About LaTeX tools that students of logic should know. *The PracTeX Journal*, 1, 2010. In Portugese. Available from http://www.tug.org/pracjourn/index.html.

[Giacomelli, 2009] Roberto Giacomelli. `calctab` package, 2009.

[Goldreich, 1991] Oded Goldreich. How NOT to write a paper, 1991. Available from http://www.wisdom.weizmann.ac.il/~oded/PS/writing.ps.

[Goldreich, 2004] Oded Goldreich. How to write a paper, 2004. Available from http://www.wisdom.weizmann.ac.il/~oded/PS/re-writing.pdf.

[Goossens *et al.*, 1994] M. Goossens, F. Mittelbach, and A. Samarin. *The LaTeX Companion*. Addison-Wesley, 1994.

[Goossens *et al.*, 1997] M. Goossens, S. Rahtz, and F. Mittelbach. *The LaTeX Graphics Companion: Illustrating documents with TeX and PostScript*. Addison-Wesley, 1997.

[Goossens *et al.*, 1999] M. Goossens, S. Rahtz, and F. Mittelbach. *The LaTeX Web Companion: Integrating TeX, HTML, and XML*. Addison-Wesley, 1999.

[Gopen and Swan, ] George D. Gopen and Judith A. Swan. The science of scientific writing. *American Scientist*, 78(6):550–558.

[Graham *et al.*, 1989] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, 1989.

[Hagen, ] Hans Hagen. Making MetaPost outlines. Available from METAFUN WIKI: http://wiki.contextgarden.net/MetaFun.

[Hagen, 2002] Hans Hagen. METAFUN, 2002. May be downloaded from the METAFUN WIKI: http://wiki.contextgarden.net/MetaFun.

[Halmos, WHEN] Paul Halmos. How to write mathematics, **WHEN**.

[Heck, 2005a] André Heck. Learning LaTeX by doing, 2005. Available from http://www.science.uva.nl/onderwijs/lesmateriaal/latex/latexcourse.pdf.

[Heck, 2005b] André Heck. Learning METAPOST by doing, 2005. Available from http://remote.science.uva.nl/~heck/Courses/mptut.pdf.

[Hurlin, 2007] Clément Hurlin. Practical introduction to METAPOST, 2007. Available from http://www-sop.inria.fr/everest/personnel/Clement.Hurlin/misc/Practical-introduction-to-MetaPost.pdf.

[Kern, 2007] Dr. Uwe Kern. Extending LaTeX's color facilities: `xcolor`, 2007.

[Knuth *et al.*, 1987] Donald E. Knuth, Tracy L. Larrabee, and Paul M. Roberts. *Mathematical Writing*. Mathematical Association of America, 1987. The TeX sources of almost the entire book are available from http://www-cs-faculty.stanford.edu/~knuth/klr.html. Digitized versions of videos of the lectures are available from http://scpd.stanford.edu/knuth/.

[Knuth, 1990] D. E. Knuth. *The TeXbook*. Addison-Wesley, 1990. The source of this book is freely available from http://www.ctan.org/tex-archive/systems/knuth/tex/.

[Krab Thorub *et al.*, 2005] Kresten Krab Thorub, Frank Jensen, and Chris Rowley. The `calc` package *Infix notation arithmetic in LaTeX*, 2005.

[Krishnan, 2003] E. Krishnan, editor. *LaTeX Tutorials* A Primer. 2003.

[Kurose, 2006] Jim Kurose. Top-10 tips for writing a paper, 2006. Available from http://www-net.cs.umass.edu/kurose/talks/top_10_tips_for_writing_a_paper.ppt.

[Lamport, 1994] L. Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, 1994.

[Lee, ] Kevin P. Lee. A guide to writing mathematics. Available from http://www.stat.ualberta.ca/~wiens/purdue1_write.pdf.

[Levin and Redell, 1983] Roy Levin and David D. Redell. An evaluation of the ninth SOSP submissions *or how (and how not) to write a good systems paper*. *ACM's Operating systems Review*, 17(3):35–40, 1983. Available from http://www.cs.umbc.edu/cra/etw98/writing-papers.pdf.

[Maltby, 1992] Gavin Maltby. An introduction to TeX and friends, 1992. Available from http://citeseer.ist.psu.edu/maltby92introduction.html.

[Oetiker *et al.*, 2007] Tobias Oetiker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl. The not so short introduction to LaTeX $2_\varepsilon$, 2007. Version 4.22, Available from http://tobi.oetiker.ch/lshort/.

[Pakin, 2005] Scot Pakin. The comprehensive LaTeX symbol list, 2005. Version 4.22, Available from http://tug.ctan.org/info/symbols/comprehensive/symbols-letter.pdf.

[Pakin, 2006] Scot Pakin. The visual LaTeX FAQ, 2006. Version 4.22, Available from http://http://tug.ctan.org/tex-archive/info/visualFAQ.

[Peyton Jones and Hughes, 1999] S. Peyton Jones and J. Hughes. `Haskell 98`: A non-strict, purely functional language, 1999.

[Peyton Jones, 2004] Simon Peyton Jones. How to write a great research paper, 2004. Available from http://research.microsoft.com/~simonpj/Papers/giving-a-talk/writing-a-paper-slides.pdf.

[Pugh:1993, 1993] Pugh:1993. Advice to authors of extended abstracts, 1993. Based on discussions among the Programme Committee of SIGPLAN'91 PLDI. Available from ftp://parcftp.xerox.com/pub/popl96/pugh/advice.ps.Z.

[Ramsey, 2006a] Norman Ramsey. Learn technical writing in two hours per week, 2006. Available from www.eecs.harvard.edu/~nr/pubs/learn.ps.

[Ramsey, 2006b] Norman Ramsey. Teach technical writing in two hours per week, 2006. Available from http://www.eecs.harvard.edu/~nr/pubs/two.pdf.

[Reckdahl, 2006] Keith Reckdahl. Using imported graphics in LaTeX and pdfLaTeX, 2006. Available from ftp://ftp.tex.ac.uk/tex-archive/info/epslatex.pdf.

[Robertson, 2008] Will Robertson. The `fontspec` package, 2008.

[Sedgewick and Flajolet, 1996] R. Sedgewick and P. Flajolet. *An Introduction to the Analysis of Algorithms*. Addison-Wesley Publishing Company, 1996.

[Senthil, 2007] Kumar M. Senthil. LaTeX tools for life scientists (BioTeXniques?). *The PracTeX Journal*, 4, 2007. Available from http://www.tug.org/pracjourn/index.html.

[Talbot, 2007] Nicola Talbot. `datatool` v 1.01: Database and data manipulation, 2007.

[Tantau, 2010] Till Tantau. TikZ & PGF, 2010. Manual for Version 2.00-cvs.

[Team, 2001a] LaTeX3 Project Team. LaTeX 2$_\varepsilon$ font selection, 2001. Available from `http://www.latex-project.org/guides/fntguide.tex`.

[Team, 2001b] LaTeX3 Project Team. LaTeX 2$_\varepsilon$ for authors, 2001. Available from `http://www.latex-project.org/guides/usrguide.tex`.

[Team, 2001c] LaTeX3 Project Team. LaTeX 2$_\varepsilon$ for class and package writers, 2001. Available from `http://www.latex-project.org/guides/clsguide.tex`.

[Team, 2001d] LaTeX3 Project Team. Configuration options for LaTeX 2$_\varepsilon$, 2001. Available from `http://www.latex-project.org/guides/cgfguide.tex`.

[Tellechea, 2010] Christian Tellechea. `spreadtab` v0.3, 2010.

[Thomson, 2008a] Paul A. Thomson. Clinical trials management on the internet — I. using LaTeX and SAS to produce customized forms. *The PracTeX Journal*, 3, 2008. Available from `http://www.tug.org/pracjourn/index.html`.

[Thomson, 2008b] Paul A. Thomson. Clinical trials management on the internet — II. using LaTeX, postscript, and SAS to produce barcode label sheets. *The PracTeX Journal*, 3, 2008. Available from `http://www.tug.org/pracjourn/index.html`.

[Trask, 1997] R. L. Trask. *Penguin Guide to Punctuation.* Penguin Books, 1997.

[van Leunen and Lipton, ] Marie-Claire van Leunen and Richard Lipton. How to have your abstract rejected. Available from http://www.cs.ucla.edu/ rupak/rejectedabstract.htm.

[Veytsman and Akhmadeeva, 2006] Boris Veytsman and Leila Akhmadeeva. Drawing medical pedigree trees with TeX and PSTricks. *The PracTeX Journal*, 4, 2006. Available from `http://www.tug.org/pracjourn/index.html`.

[Voß, 2008] Herbert Voß. *Tabellen mit LaTeX.* Lehmanns / Dante e.V., 2008.

[Voß, 2009] Herbert Voß. Math mode — v. 2.43, 2009. CHECK THIS: Avalable from `ftp://cam.ctan.org/tex-archive/info/math/voss/mathmode/Mathmode.pdf`.

[Walsh, ] Toby Walsh. How to write a thesis. Available from `http://www.cse.unsw.edu.au/~tw/thesis2.ppt`.

[Write, 2008] Joseph Write. `siunitx` — A Comprehensive (SI) units package, 2008.

[Zorn, ] Paul Zorn. Tips on writing in mathematics. Available from `http://www.stolaf.edu/people/zorn/prooftips.pdf`.

# 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 TeX and to Leslie Lamport for writing 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 thank Luca Mercriadri, 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 LaTeX and friends, all those who have supported LaTeX and friends, and all who have answered all my 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, Taco Hoekwater, Thomas Esser, Ulrike Fisher, Victor Eijkhout, Vincent Zoonekynd, Will Robertson, and all the many many others. Without them, the TeX community would have been much worse off.

# Colophon

This document was typeset using the `pdflatex` program using the '`online-not-printed`' option of a personal class file — this results in more colours which may not always print as nice as the `printed-not-online` option. Chapters have been used to prepare presentations using the `beamer` class. Except from changes in class options, all typesetting was done from the same source files. The main text was typeset using the `book` class with a `11pt` option. The size of the document was set using the `a4wide` class, which gives you slightly narrower margins than the `a4size` option of the `book` class. The `fourier` package was used to set the text font to *Utopia Regular* and the math font to *Fourier*. The monospaced font was typeset with the `beramono` package with '`scaled=0.83`' option. The `amsmath` and `amssymb` packages were used to help typesetting the mathematics. The coverpage is drawn with the help of a little `tikz` program. The chapter style is based on a design from Vincent Zoonekynd, using Euler chapter numbers instead of regular numbers.