

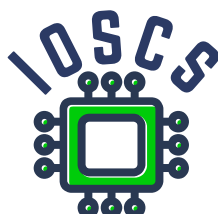
Mendel University in Brno

Open Source Tools for Text Processing

Study text

Jiří Rybička
Mendel University in Brno

Project: Innovative Open Source Courses
for Computer Science Curriculum



24. 6. 2022



Co-funded by the
Erasmus+ Programme
of the European Union



West Pomeranian
University of Technology
Szczecin



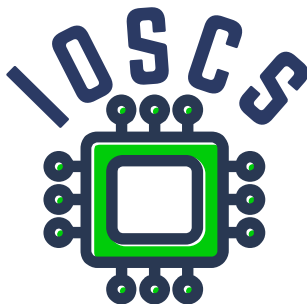
UNIVERSITY
OF ŽILINA

Mendel
University
in Brno

Reviewer: Doc. RNDr. Ján Buša, CSc., Technical University of Košice, Slovakia
Project: Innovative Open Source Courses for Computer Science Curriculum ©
Mendel University in Brno, Zemědělská 1, 613 00 Brno, Czech Republic
ISBN 978-80-7509-880-1 (online ; pdf)
DOI <https://doi.org/10.11118/978-80-7509-880-1>



Open Access. This book is licensed under the terms of the Creative Commons Attribution-ShareAlike 4.0 International License, CC BY-SA 4.0 (<https://creativecommons.org/licenses/by-sa/4.0/>)



This material teaching was written as one of the outputs of the project “Innovative Open Source Courses for Computer Science Curriculum”, funded by the Erasmus+ grant no. 2019-1-PL01-KA203-065564. The project is coordinated by West Pomeranian University of Technology in Szczecin (Poland) and is implemented in partnership with Mendel University in Brno (Czech Republic) and University of Žilina (Slovak Republic). The project implementation timeline is September 2019 to December 2022.

Project information

Project was implemented under the Erasmus+.

Project name: “[Innovative Open Source courses for Computer Science curriculum](#)”

Project nr: [2019-1-PL01-KA203-065564](#)

Key Action: [KA2 – Cooperation for innovation and the exchange of good practices](#)

Action Type: [KA203 – Strategic Partnerships for higher education](#)

Consortium

ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY W SZCZECINIE

MENDELOVA UNIVERZITA V BRNĚ

ŽILINSKÁ UNIVERZITA V ŽILINĚ

Erasmus+ Disclaimer

This project has been funded with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Copyright Notice

This content was created by the IOSCS consortium: 2019–2022. The content is Copyrighted and distributed under Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).



Co-funded by the
Erasmus+ Programme
of the European Union

Preface

This book contains study material for the course Open source word processing tools intended for education at the university level. It is the result of a project dealing with the application of open source tools in teaching. It shows that even for demanding publications there is no need to use expensive licensed software, and it is even possible to efficiently create documents that are difficult to implement in other environments or with a high proportion of manual work.

The text emphasizes an efficient approach to document processing – the wide use of so-called structural (semantic) markup that allow separating the document from its visual form, thus achieving greater flexibility and reusability of the processed material.

The technological tool is the Open Source system $\text{X}_{\text{Y}}\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ widely used in the scientific and professional field, in education and in selected publishing houses. It is a principle whose mastery brings significant advantages to users when processing professional texts, such as various seminar and final theses, articles, proceedings and other publications.

Together with the individual elements of the $\text{X}_{\text{Y}}\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ system, the book also contains an explanation of the necessary typographical principles and rules applied in common types of documents. It thus brings a complete view of document processing.

Acknowledgments

At this point, I would like to express my gratitude to doc. RNDr. Ján Buša, CSc., for many valuable comments and suggestions on this text.

Contents

1	Introduction	8
2	Document and method of its processing	9
2.1	Principle of document elements	10
2.2	Identification of elements in the document	10
2.3	Representation of elements – typographic design	11
2.4	Representation of elements – implementation of typographic design . . .	12
2.4.1	Structural markup	12
2.4.2	Software	12
2.5	Technological principle of systems based on the \TeX	13
2.5.1	Basic principle of \TeX	13
2.5.2	Extensions, distributions	14
2.5.3	Working with the system, source text, commands	15
2.6	Command types, groups, environments	16
2.7	Custom definition	17
2.8	Technological principle based on systems in office packages	18
3	Basic document parameters	19
3.1	Book font – types	19
3.2	Body font and its parameters	19
3.3	Document in printed or electronic form	22
4	National environment and special characters	24
4.1	Input text encoding	24
4.2	Locale setting	24
4.3	Hyphenation algorithm and its settings	25
4.4	Special characters and their solutions	26
5	Paragraph typesetting, algorithms, parameters	29
5.1	Typographic measuring systems, lengths, length registers	29
5.1.1	Paragraph	30
5.2	Body text typesetting parameters	30
5.3	Other paragraph document elements	34
6	Mixed typesetting	36
6.1	Use of additional typeface	36
6.2	Emphasizing	36
6.3	Technology – colors, models, definitions	39

7	Document division	41
7.1	Heading systems	41
7.2	Initials	42
7.3	Table of contents creation	42
7.4	Technology – numbering, counters and references	44
8	Pages	46
8.1	Page elements	46
8.2	Relationship between paragraph and page break	46
8.3	Page headers and footers	46
8.4	Footnotes	47
8.5	Marginalia	48
8.6	Page and paper	49
8.7	Special pages design	50
9	Mathematical and similar expressions	52
9.1	Elements of expressions	52
9.2	Inserting of expressions into the document, references	57
10	Tables	59
10.1	Table types	59
10.2	Ways to align tabular data	59
10.3	Inserting tables into document	62
11	Image material and graphics	64
11.1	Types of images and their sources	64
11.2	Graphic elements in the document	69
11.2.1	Special fonts characters	69
11.2.2	Typesetting elements and operations with them	69
11.2.3	Typesetting principle and \TeX modes	70
11.2.4	Boxes	71
11.2.5	Affecting the base typesetting method	75
11.2.6	Graphics operations	76
11.3	Inserting images in a document	77
11.4	Creating pictures in the system – environment <code>picture</code>	77
12	Document as a whole	81
12.1	Pages arrangement	81
12.1.1	One-sided or two-sided?	81
12.1.2	Document arrangement	81
12.2	Moving of information	84
12.2.1	Fragile and robust commands	84
12.2.2	Index	84
12.2.3	References – bibliographic citations	86
12.3	Production of a physical document	89
12.3.1	Imposition pages for printing	89
12.3.2	Overview of bookbindings	91

13	References for further study	94
A	Overview of elements used in mathematical environments	96
B	How to install and use a working distribution	99
B.1	User Support	99
C	For inspiration – how this publication is typeset	101

Introduction

Computer word processing has undoubtedly been for a long time one of the most frequented applications. Best practices and rules can save a lot of time and resources in preparing and reusing texts to the vast majority users. The way to this goal is to get acquainted with two areas – typography (defines the general requirements for the appearance of the document) and the relevant computer applications (implements typographic requirements). Although it is a mass one application, in our experience, relatively few users are aware on the effective preparation and further processing of documents of various types and purposes.

This text aims to present the two basic areas needed for quality and efficient word processing: typography and selected technology. In each chapter, therefore, we encounter a section stating the problem in typographic level and this will be followed by an explanation of the technical elements and tools of the given system. The typographic part is more general and applicable with any technology, but the implementation of the document is only dependent on selected tools and their possibilities. In this text we deal with technology based on the typographic system \TeX , which is freely available and is available on all operating platforms. In addition, the link to fundamentally different technology of word processing by the respective programs available as part of office packages.

Mastering any program system involves a certain complexity especially the need for systematic acquaintance with the basic principles of activity and with all the essential services that the program offers. It also assumes that a certain time investment, which starts to pay off after a certain time. If it will show in the meantime that the services of the program do not meet the needs of the user, it is possible state that this investment was not good.

In the case of systems based on the \TeX principle, the situation is similar, but our experience suggests that although the time investment seems *somewhat* larger than in the case of other programs of a similar class, its return tends to be *substantially* larger. The \TeX system is so specific that it does not have significant competition even today. In this respect, it brings unique effects.

This brief text dealing with the \TeX extension of $\text{X}_{\text{\LaTeX}}$ cannot contain everything that would be appropriate for the practice to know. However, it forms an introduction. The basics can easily be continued in further study, both from the literature mentioned in the final list, as well as partly from electronic sources on the Internet or documentation texts in the distribution used.

Document and method of its processing

The term **document** will be used to define the logical and physical unity of content and forms. From this point of view, the document is, for example, a business letter, invoice, business trip report, annual report, but also a spreadsheet, because it also contains information for formatting of the content (numbers and expressions), ie form.

In general, we need to manipulate both in any document processing parts – both in content and form. We will not discuss the content (with some exceptions), it is the subject of other areas, we focus mainly on form and its efficient processing.

Work on the document can be divided (also according to long-term historical experience) between three roles:

- **Author** – the result of the author’s role is the content part of a document. The author is an expert who decides what will be in the document, what is important, how the text will be formulated and what information will be presented in tables, mathematical expressions and figures or diagrams.
- **Designer** – the role of the designer presupposes the design of the document, according to the intended purpose (printed document, electronic document or part of a proceedings or book, part of another larger unit). The designer is an expert in the field of typography or graphic design, decides what parameters of elements will be optimal for a given document, so that the reader perceives and processes it as best as possible.
- **Typesetter** – the role of the typesetter represents the technical implementation of everything what the author and editor created, using appropriate technology. The typesetter is a technical expert – he controls the relevant software, he can apply suitable functions to the document so that its implementation and any subsequent modification is as effective as possible.

These three roles can be played by one physical person, which is very often the today’s case. In earlier times, when documents were created only in professional factories, it was strictly different people who were specialized in their share of the document, which ensured a certain standard quality of the final work. Nowadays, it does not require extensive knowledge of the technology of the so-called hot typesetting from metal letters, the computer all technical elements implement today, but there is still a need to deal with typographic rules and the technical possibilities that we have at our disposal. That is the aim of this text.

Although we take ourselves into all three of these roles, we should always be very aware of which of them we are currently in when processing a document, and we should not combine their activities together. Concentrating on one of these roles always enables its quality application, and thus the possibility of obtaining an optimal result. It is also

possible to pass one of these roles to an expert, thus avoiding a document that is imperfect or even unacceptable.

2.1 Principle of document elements

Each document has its own logical structure. We can divide it into sections to which a certain meaning can be assigned (ordinary paragraph, title, footnote) – so-called **document elements**. This classification is very much related to the content of the document, so only the author of the document can determine it without errors. For example, the author determines (as already mentioned) where each paragraph ends or which term is important and which phrase has what meaning. Of course, in some specific simple cases, it is possible to arrange for the elements to be determined by the document designer (or editor, proofreader, typesetter), but this is an unreliable and sometimes dangerous method, as it can confuse meanings.

In addition to its content, each document element can also have a function within the document. Most often, it is a visual function so that the reader can clearly distinguish it from other elements, but this does not have to end there: elements can appear, for example, in the table of content of the document, in page headings, in registers or can be processed by external programs, inserted into various reports, databases, etc.

2.2 Identification of elements in the document

An irreplaceable function of the author of the document content is (as already derived) also the determination of the document elements, which can be done in different ways, especially depending on who represents the other two roles in editing the document.

Inserting information about the type of document element is most often done using a structural markup (see below). However, it is also possible to use any other method of marking (for example, in pencil on the printout of the text content of the document).

What elements are typical? Almost every document will have a paragraph of the body text as one of the elements. In the same way, we can often see other elements: headings of various categories, enumerations and lists, footnotes, table fields, captions of figures and tables, mathematical expressions, quotes, etc. It would seem that on this is nothing new – after all, most different software systems have their solution available from the usual menus for the mentioned documentary elements. But we will go further: the listed general elements usually need to be supplemented by other, specific for the given document, for which general offers are no longer enough.

An example is a business letter. Its typical elements are shown in tab. 2.1.

Tab. 2.1: *Elements of a business letter*

<i>Element name</i>	<i>Description</i>	<i>Note</i>
Address	recipient's address	contains 4 parts – lines of address
Appeal	appeal data	contains 3 parts – id, place, handles
Subject	standard letter element	single line text
Salutation	salutation of recipient	it can have a variant for men and for a women
Text	one paragraph of letter text	can be repeated as needed
Greeting	final greeting	contains the person's name and function
Attachment	reference to attachments	it need not be specified if there are no attachments

As you may argue, this table is not the only possible solution to the letter. You can certainly add more elements, or, conversely, remove or change the meanings of existing ones. This is not important at the moment, but rather indicates the need to determine these elements according to the immediate need and not to rely on someone (or any software) to do it for us.

2.3 Representation of elements – typographic design

The second step is to change the role from author to designer. Now we have to assign to the identified elements their appearance and possibly also functions within the document. We will start from the previous table and add a suitable form while respecting the typographic rules and graphic design. In many cases, we can adhere to two design principles:

- Principle of unity
- Principle of contrast

The principle of unity states that elements of a document of the same meaning must have a *uniform appearance*. The principle of contrast adds: elements of a document of different meaning must have a *sufficiently different appearance*. We achieve uniformity through structural markup, to achieve contrast we apply a number of different typographic principles (application of sizes, colors, font types, font shapes, etc.).

Figure 2.2 shows what one of the typographic design options might look like.

Tab. 2.2: *Graphic design of letter*

<i>Element name</i>	<i>Parameters</i>
Address	body font, 12 pt, body font shape
Appeal	body font, 10 pt, body font shape
Subject	body font, 12 pt, bold
Salutation	body font, 12 pt, body font shape
Text	body font, 12 pt, b. shape, baseline skip 15 pt, align left, spacing 8 pt
Greeting	body font, 12 pt, align left, spacing 20 pt
Attachment	body font, 10 pt, itemize, bullet

In addition to the visual appearance of the individual elements, the designer also determines the page parameters (letter head, shape of the first page, shape of any continuation pages etc.).

2.4 Representation of elements – implementation of typographic design

After connecting the elements of the document with their typographic design, we get a detailed overview of the required final form of the document, we can now process the implementation using the tools of a specific program.

All word processing programs have a common general principle. They create a document as a conjunction of content and **formatting tags**, which represents the connection between content and form (in the sense of our definition of document). These tags (or also commands, processing commands) can be divided into two categories:

1. **Visual tags** – define the appearance of the relevant element. Examples: italic font shape; eight-point size adjustment; creating a vertical gap of 20 points; justified align of a paragraph.
2. **Structural tags** – define the meaning of the element. Examples: second level heading; footnote; table field with text content; citation from the literature.

At first sight, it might seem that visual tags are more important: we are trying to implement the document in the form that we derived according to the typographic rules for each element. However, we must be aware of one important fact in this context: the structural mark carries information about the meaning that only the content author can add to the document – just as only the author determines the individual elements of the document. Usually, we assign structural tags to the individual elements found in a document, which inserts author's information into the computer document.

2.4.1 Structural markup

If a structural tag is essential, then it is quite obvious that the technological goal is to achieve **structural markup**, i.e. the use of structural tags to identify individual physical elements of a document. If the used software system allows to mark the document exclusively with structural tags, it is purely structural markup and the obtained document is from the technological point of view then arbitrarily processable and reusable. However, some systems are not able to include all document settings in structural tags.

However, the structural tag alone is not sufficient to create the document. It is still necessary to have its visual appearance. For efficient document processing, it is essential how *assigns* a set of visual parameters to the appropriate structural tag. It can be an “inseparable couple”, where the visual parameters are permanently attached directly to the structural tag, or the visual definition *can be separated* from the place where the structural tag is used, even in a file outside the document. In the second case, it is possible to *replace* the visual definition of a certain structural tag, which allows *without editing the document* to change the overall appearance or behavior of any elements. In this case, a very efficient and flexible processing of the document can be achieved.

2.4.2 Software

The concepts of word processing software are very different, but we can detect at least three basic categories:

- programs originally modeling a typewriter, suitable for preparing source codes of programs, HTML documents or similar files; they can be called **program editor** (examples: vi, vim, NotePad, PSPad, KEdit, Emacs and many more),
- programs originally modeling a typewriter, but later enriched by various functions suitable for the preparation of text documents (various font types and typefaces, work with paragraphs, pages and document units); they are called **text editor** or **word processor** (examples: WordPad, Word, OO Writer, etc.),
- programs modeling a book typesetting, or at least some parts thereof; the term **DTP system** (Desk Top Publishing) is used for them. Typical representatives are InDesign, QuarkXPress, \TeX .

None of the software systems can do everything, so it is necessary to choose a suitable alternative for each application, even between systems in the same category. We also need to know at least a general definition of the functions of individual systems in order to be able to choose the appropriate variant.

It follows from the above analysis that the crucial feature that we will look for and use in word processing software is the possibility of using structural markup. In this context, it is no longer necessary to consider the category of program editors (we will use them in another role), but programs that insert tags in documents are in the category of word processors and DTP systems.

2.5 Technological principle of systems based on the \TeX

\TeX based systems belong to the category of computer models of book typesetting, but they are characterized by completely different properties compared to other representatives of this category.

2.5.1 Basic principle of \TeX

The original idea of the \TeX system creator, Donald E. Knuth of Stanford University, was a computer model of the work of a quality typesetter for mathematical typesetting. This has always been the most demanding part of creating professional books, including both precise mastery of the craft and the ability to bring a sense of result to the rate. The name \TeX thus symbolizes the three initial letters $\tau\epsilon\chi$ of Greek word “ $\tau\epsilon\chi\nu\omicron\lambda\omicron\gamma\iota\alpha$ ” which means technology and at the same time art, so it reads “tech”.

At the time this system was created (first attempt 1978), completely different computer technologies were available than today. The basic principle also follows from this – the author (or maybe also typesetter) writes his text, in which he inserts commands for typesetting. This source is then analyzed and processed into the resulting material typesetting in output pages.

The basic principle of the system is shown in fig. 2.1.

As can be seen here, the core of the system is the processing of the source text, which is accompanied by information about predefined commands, typesetting styles, fonts used, etc., the main result of the translation is typeset material in DVI (DeVice Independent) format, from which can be obtained for example PostScript format. The main output is also more recently in PDF format. In addition to the result, the compiler writes

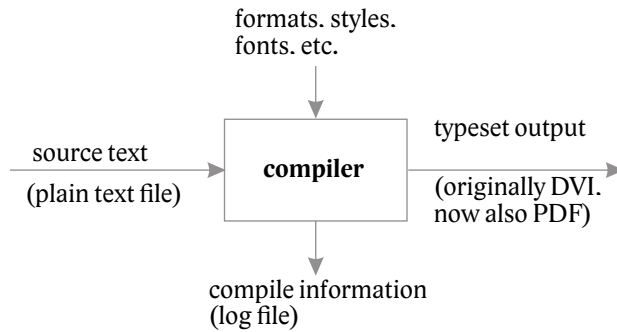


Figure 2.1: Basic principle of the \TeX

a comprehensive report, from which can be read all relevant information, possible errors or deficiencies of the typesetting process. The core of the system is usually surrounded by a large number of other supporting programs – program editors for source text or automated source text generators, programs for subsequent processing of typeset output, programs supplying the necessary components (styles, fonts, images), etc.

D. Knuth focused mainly on high quality output. The system follows all the details of the work of a qualified typesetter very precisely and allows you to apply typographic principles created over several centuries. The specific font Computer Modern designed for ideal math typesetting is also as a part of system. In this respect, the system has no competition to date.

The second crucial feature is the ability to change the command system in almost any way to the user’s needs – there is a way to create any new command or change the behavior of an existing command, including the ability to change the perception of individual symbols written in the source text. This is also a unique approach.

Creating any commands can be used to create purely structural documents. The new commands can be designed as structural tags and assigned almost any visual form. In addition, the definition of the visual appearance of tags can be separated from the document into style files, document classes and similar external definitions, thus gaining the ability to efficiently reuse identical structural tags in another visual and even functional variant. In the field of word processing systems, there is no alternative that can do more than this language.

2.5.2 Extensions, distributions

\TeX itself has dozens of commands that are able to control the typesetting process in great detail. However, they are quite complex for the average user and are often unnecessarily detailed. Therefore, a number of **extensions** have been created to offer a more user-friendly environment, but at the same time take advantage of all the benefits and excellent features of \TeX . One of the best known and most used extensions is \LaTeX , originally written by L. Lamport (1984). It provides a number of commands that greatly facilitate the creation of common documents (reports, articles, books, letters, presentations).

However, the development continues to this day and a number of other extensions are emerging. One of the most prominent is $X_{\text{}}\TeX$ allows word processing in many

languages, typesetting from left to right and right to left, the use of system fonts, etc. and $X_{\text{L}}\text{L}\text{A}\text{T}\text{E}\text{X}$ which extends $\text{L}\text{A}\text{T}\text{E}\text{X}$. Recently, also other extension – the $\text{Con}\text{T}\text{E}\text{X}\text{T}$ system is a completely different view of the way the document, and significant is also Czech developer Petr Olšák’s $\text{Op}\text{T}\text{E}\text{X}$. The aim of all extensions is mainly to adapt the system to current needs and possibilities while maintaining the basic principle, which allows to obtain precise and typographically high-quality documents.

The collection of several compiler variants, many support programs, available styles, fonts and other necessary components form together the so-called **distribution**. There are several distributions, one of the most common being $\text{T}\text{E}\text{X}\text{Live}$. It is characterized not only by the extent and complexity, but also by the fact that all components are available free of charge. It can be installed from the distribution DVD or directly from the Internet, from www.texlive.org. All common operating systems (Windows, Linux, MacOS) are supported. The distribution is updated annually and its development is taken care of by a large team of people.

Another well-known and widespread distribution is $\text{Mik}\text{T}\text{E}\text{X}$, which specializes in Windows operating systems.

In addition to installation on a local computer, it is also possible to use Internet access. At <https://tex.mendelu.cz> there is a web interface, with which it is very easy to type the source text and get the final output in PDF or PostScript format. Overleaf is also a well-known project, enabling dynamic viewing of typed output after each change of source text.

In the whole next text we will describe the work with the extension $X_{\text{L}}\text{L}\text{A}\text{T}\text{E}\text{X}$ and the output format will be PDF. This is an alternative that is very widespread and covers a large number of user needs. Because $X_{\text{L}}\text{L}\text{A}\text{T}\text{E}\text{X}$ is based on $\text{L}\text{A}\text{T}\text{E}\text{X}$ system definitions, we will talk about both systems.

2.5.3 Working with the system, source text, commands

After installing the selected distribution, the user has the selected compiler at his disposal, he can choose any program editor to create the source text, and he uses a suitable browser to display the output. Working with the system consists of a repeated sequence of three steps¹:

1. creating or editing source text in some program editor (usually we use freely available editor – PSpad, Notepad++, vim, joe, etc.),
2. compilation of the source text by the selected compiler into the selected output format (for example, the xelatex compiler with output to PDF format),
3. viewing the result by the output format viewer (in the case of the PDF format, for example, the generally freely available Acrobat Reader can be used, Ocular or other alternatives in all used operating systems are very advantageous).

Compared to InDesign, for example, $\text{L}\text{A}\text{T}\text{E}\text{X}$ differs mainly from “non WYSIWYG”². However, it is necessary to say that this feature is very questionable – the so-called work in the preview (ie. the situation where we directly edit in the final form of the document)

¹Some user interfaces ($\text{T}\text{E}\text{X}\text{works}$, $\text{T}\text{E}\text{X}\text{studio}$, Overleaf) implement these steps in one place.

²WYSIWYG = What You See Is What You Get, ie the user works directly with the resulting document shape.

has a number of major disadvantages, starting with a very problematic orientation in the final typesetting (many things are too small to be visible on the computer screen), despite the inability to optimize and automate a number of processes to the very difficult manipulation of some elements – for example special characters at a plain typesetting. Thus, in the case of \LaTeX , this is a disadvantage that is perceived mostly only by a beginner who has not yet recognized the immense power of typeset automation and extensive possibilities of precise output control.

2.6 Command types, groups, environments

The key part of working with the \TeX system is therefore the editing of the source text. Throughout the following text, we will focus on the commands placed in the input text material, their concept and possibilities.

The compiler **command** can have three forms:

1. form so-called **active character** – the command consists of one character, e.g. \$, % or &;
2. form of one-character command – command starts with backslash and follows one non-alphabetic character, e.g. \; or \=,
3. form of word command – command starts with backslash and then follows sequence of alphabetic characters, e.g. \item or \footnote. After word command have to be a delimiter – any non-alphabetic character or spaces or line breaks. If the delimiter is a group of spaces or the end of a line, that delimiter is absorbed and does not reach the output. This behavior allows you to effectively insert word commands into the text without accumulating parasitic spaces in the output.

Commands can have **parameters**. Parameters can be mandatory (usually written in braces {...}) or optional (always written in square brackets [...]). The optional parameter can be omitted, then its parentheses are also omitted. The order of mandatory and optional parameters is determined by the command definition. Example: `\makebox[2em][r]{kk}`

Commands have different **scopes**. Some commands have only a local effect (they only operate where they are written), some have a completely global effect (regardless of where they are mentioned, they will affect the whole document). However, most commands apply to a specific bounded area called **group**.

We can define the group in the document in two ways:

1. by curly braces {...}
2. by so-called **environment**, i.e. the part of the document bounded by the commands `\begin{name}... \end{name}`

Groups can be nested as needed, but groups cannot be intersected, i.e., an inner group cannot be closed later than an outer group.

The command scope begins at the place where is written and ends when the relevant group is closed.

2.7 Custom definition

Commands that are available and already predefined can be enriched with more, which the user can create himself. Custom definitions are relatively simple. Definitions can be used to create new commands and change the definition of existing ones.

It may be strange at first glance that we are currently creating and modifying commands without presenting predefined commands, but the concept of adapting the system to individual documents and specific user needs requires that we go hand in hand with both groups at the same time.

There are several ways to create your own command, including parameters:

- command `\def` and its variants, this method can be used both for new commands and for redefining existing ones;
- command `\newcommand` to create a new command;
- command `\renewcommand` to redefine an existing one;
- command `\newenvironment` to create a new environment;
- command `\renewenvironment` to redefine an existing one.

Examples

Ex. 2.1: `\newcommand{\greeting}{Good morning!}` – the command only prints the text “Good morning!”.

Ex. 2.2: `\def\salutationman#1{Dear Mr. #1}` creates command `\salutationman` with one parameter, into which we can enter the name. Example of use:
`\salutationman{Fisher}` produces the text “Dear Mr. Fisher”. The maximum number of parameters can be 9. The places where the text of a certain parameter is inserted in the definition are indicated by the # character and the corresponding number.

Ex. 2.3: `\newcommand{\salutationman}[1]{Dear Mr. #1}` – the same as above.

Ex. 2.4: `\renewcommand{\salutationman}[1]{Good morning, Mr. #1}` – changes the definition of a previously created command.

Ex. 2.5: `\def\finalgreeting#1#2{Regards \\ #1 \\ #2}` – creates a command (or changes the definition of an existing one) `\finalgreeting`, which prints the text “Regards” and under it two lines containing two parameters.

Principle of structural markup of documents

Since we can produce any custom commands in the $\text{X}_{\text{L}}\text{A}_{\text{T}}\text{E}_{\text{X}}$ system, we can only compile the entire document using structural tags, the visual appearance of which will be included in the definitions. We can modify these definitions later at any time without having to change the document.

Example of a structurally marked document

```
\address{Dear Mr.}{Jack Newman}
      {Wood avenue 654/11}{67411}{Litchfield}
\refer{13-21/2}{Hutchinson}{Claire Smith}
\subject{Payment advice}
\salutationman{Newman},
\text{we are pleased to inform you that the goods you provided
us were sold according to your instructions, so we will send
the payment to your account.}
\greeting{Anete Garmins}{sales department}
\attachment{Confirmation of the completed transaction}
```

The text does not need comment, the meanings of the individual parts are clear based on the appropriate choice of command names. At the same time, however, we see that there is not any mention of the visual form of formatting in the document. This is concentrated in the definitions of the individual commands.

2.8 Technological principle based on systems in office packages

The term **office package** usually refers to a set of programs used in common user practice and covering the most common user needs. Word processing programs are an undoubted and indispensable part of such packages. A representative of free software is Writer from Open Office or Libre Office, and it is an alternative to programs from licensed office suites.

The Writer program belongs to the category of word processors. The basic different feature compared to T_EX-like systems is the interactive way of marking a document. Most users use only visual tags in this way, which are placed in a well-accessible form of user interface.

Nevertheless, even this program has the ability to create and use structural marking of documents relatively effectively. The base prop is **formatting style**, which is a *named group of visual parameters*. The style name represents the structural information, the set of visual parameters then define the appearance of this tag. If the structural mark is assigned to the selected part of the document, then by changing any visual parameter of the given style it is possible to automatically reformat all parts to which the style is assigned.

You can also create links between styles – with this method it is possible to *promote* the change of one visual parameter to several child styles, and thus ensure the logical consistency of the entire document.

Unfortunately, styles do not cover all document parameters. These are most often styles applicable only to paragraphs or parts of paragraphs. Other settings are firmly linked to the document and cannot be changed effectively (page elements). However, all settings can be concentrated in a template, based on which new documents are created.

Basic document parameters

When determining the design of a document, it is necessary to choose certain parameters, from which other parts are then derived. One of the most important parameters is the font used and its basic size.

3.1 Book font – types

All the fonts we can see in the computer world can be divided into two major categories – **typewriter** and **book**. Typewriting was created as a technical prop for typewriters, the basic purpose of which was to replace handwritten documents in a negligible number of copies and a relatively small scale. Due to their simplified design, typewriters did not allow the use of fonts with high aesthetic and technical quality, so nowadays there is no reason to use such fonts for ordinary documents.

We will deal only with book fonts, the basic feature of which, compared to typewriter, is the different width of individual characters. This also entails a completely different way of processing documents than on typewriters. One of the deepest mistakes we can make, then, is that we apply the rules and principles that apply to a typewriter to a document in book type.

Book fonts can be further divided into approximately three major categories:

- Antique fonts (serif) – are intended for printed documents and represent a basic branch in typographic development since the 15th century.
- Grottesque fonts (also sans-serif) – in printed documents they are considered as an additional (headings, captions, subtitles, etc.), in electronic documents their simplified drawing is used for a clearer display.
- Other fonts – for special occasions (also accidental). This includes various decorative, calligraphic, handwritten, angled and other fonts. They are very rarely used in common documents.

3.2 Body font and its parameters

One of the most basic parameters of documents is used **font** – its type, face and size. The whole document is usually created in one font type (combinations of types are sometimes just an unnecessary complication in design). We will call this font **body font**. The most commonly used is the so-called **plain face** of this type, and also the **base size**.

Different typefaces and font sizes are used in the document for different elements, but the font type usually remains the same for the entire document.

The $\text{\LaTeX}/\text{\XeLaTeX}$ system has the option of setting the body font of the document so that we don't have to worry about it anywhere else. This concept is enriched by the relatively well-thought-out possibility of changing sections and degrees so that the document as a whole looks good. However, this is balanced by the fact that such a central choice cannot be completely arbitrary, certain options are predefined, from which it is then possible to choose.

Implementation in the \XeLaTeX system

The document in the \XeLaTeX system has its own structure – it consists of an introductory command introducing the so-called document class, followed by a preamble where various definitions of global commands and settings and styles can be added (in system terminology they are also says packages), the preamble is followed by the document environment. In simple terms, we can show the structure by the following scheme:

```
                Struktura dokumentu
\documentclass[options]{class}
...document preamble; various definitions are placed here
\usepackage{style} % link to package
\begin{document}
...document body; the text is written here
\end{document}
```

options – an optional parameter allows you to specify some global settings in a given document class: `options twoside`, `options oneside` sets a two-sided or one-sided document, `options 11pt`, `options 12pt` set the base font size to 11 or 12 old English typographic points. The default is one-sided document and default font size is 10 points.

class – defines the document type. The predefined classes are: `article`, `book`, `report`, `letter`, `slides`.
The class can be made by the user according to their needs. It is a file with extension `.cls` which contains command definitions.

style – style (package) name with command definitions. There is a plethora of these packages available. For the typesetting in national languages, we need at least `polyglossia` package to define all the languages used in the document. The package can be created by the user, it is a text file with the extension `.sty`. All structural tags needed to write a given document type can be created in the document class definition and in the document style definition.

`%` – the `%` character introduces a comment, the translator skips everything up to the end of the line, it does not even take into account the end-of-line character itself, which under normal circumstances is interpreted as a space in the typesetting.

By default, the base font has a type derived from Knuth's Computer Modern font. It is a font very characteristic of systems based on the \TeX principle, because this font has always been available in every distribution, including national modifications. It is a

so-called static serif, which is suitable for professional documents, it has standard faces, a sans-serif variant, typewritten font and fonts for mathematical typesetting. It is optimized in several ways for typesetting by the TeX system and its extensions.

However, it is not suited to typeset all documents in the same typeface. There are a number of quality fonts suitable for professional texts, books of various specializations and documents of different purposes. In the XeLaTeX system, we can use any font installed on the given computer. This is arranged by the package `fontspec`, which is introduced together with other props by the package `xltxtra`. In the following example document, the base font Lido STF, base size 12 point is selected:

Example of document

```
\documentclass[12pt]{article}
\usepackage{xltxtra}           % document XeLaTeX, UTF-8 encoding
\usepackage[a4paper]{geometry} % paper A4
\usepackage{polyglossia}      % language switching
\setdefaultlanguage{english}  % default language is English
\setotherlanguage{slovak}     % alternative language is Slovak
\setmainfont{LidoSTF}         % base font type is Lido STF
\textwidth=16cm               % text width is 16 cm
\begin{document}
English is set as a first language. If we want to switch
to Slovak, which we also have available, we will write
a command\selectlanguage{slovak} a nasledujúci text
môžeme písať v slovenčine.
\end{document}
```

After compiling, we get approximately the following result:

English is set as a first language. If we want to switch to Slovak, which we also have available, we will write a command a nasledujúci text môžeme písať v slovenčine.

For practice:

1. Create a document on B5 paper with the body 11-point Bookman font. Write a few lines of text in your mother language and add at least three lines in English. Switch the language accordingly, watch for possible hyphenation.
2. Try typing the same document in Times font type and see the difference between the space occupied by Bookman and Times.
3. Typeset the same text on A5 paper in a 10-point font with the default font.
4. Experiment with the width of the text – try, for example, what the output will look like in a narrow column (around 5 cm). Observe hyphenation in the languages used.

3.3 Document in printed or electronic form

We can detect a certain difference between the printed and electronic final form of the document. While printing on paper is usually performed with a raster density of at least 600 dpi (but also in the conditions of ordinary office practice significantly more), the display (projector, mobile phone, etc.) is at a much lower density (around 100 dpi). It clearly follows that we cannot expect high-quality rendering of details in a document intended only for display on the screens of individual devices. We have to choose such tools that cope clearly with the given lower display density.

The most common and obvious change is choosing the appropriate document font. While a paper document will typically have one of the serif types as the base font, electronic display requires a font without small details, hair strokes, or complex drawings that may not display perfectly enough, especially in smaller sizes. We use sans-serif fonts, taking into account the expected size and density of the display.

The second significant change is also the density of embedded raster graphics. It is not necessary to prepare image information in a density that significantly exceeds the expected display – this increases (with quadratic dependence) the size of the files and complicates the transfer and any other processing.

If we are considering a publication in both printed and electronic form, we will advantageously use alternative visual definitions of structural marks and prepare a double document format. Of course, this also requires the preparation of raster images inserted into a double-density document; vector images that do not need any change are very advantageous.

Printed publications also often reduce color requirements for cost-saving reasons: while color publications are as “expensive” as black-and-white images for electronic publications, colors pay many times higher production costs for printing. Alternative definitions of structural marks can also ensure that the document is always displayed in the appropriate visual form.

Implementation in the $\text{X}_{\text{Y}}\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ system

How to create and apply alternative definitions of structural marks? We will present the possibilities usable in various situations, but meeting the basic requirement: according to some central choice (in one place of the document) it is possible to switch the overall appearance of the document for the required type of output (paper, screen, color ...).

1. The first (simplest and most general) option is to create different style files for different types of output, e.g. `letterprint.sty` and `letterscreen.sty`. In both of these files, there will be definitions of the same structural commands, but each with respect to the type of output. For example, in the file `letterprint.sty` there will be a base font definition `\setmainfont{Constantia}`, while in the file `letterscreen.sty` will be `\setmainfont{Verdana}`. You can change the output by changing the command in the document preamble:

```
\usepackage{letterprint} or \usepackage{letterscreen}
```

2. If for some reason we want to create only one style file for a given document and there are not many definitions of structural tags, then we can create alternative def-

initions as part of a distinguishing statement (or set of distinguishing statements). The basic concept can look like this:

— Alternative command definitions —

```
\def\topaper{\setmainfont{Constantia}
             \def\subject#1{\bfseries #1}} ... }

\def\toscreen{\setmainfont{Tahoma}
              \def\subject#1{\bfseries\color{blue}#1}} ... }
```

The distinguishing commands here are `\topaper` and `\toscreen` – we will write them as needed (probably in the preamble) in the document and the rest of the structural marks will receive alternative definitions. As can be seen from the example, the body of distinctive statement definitions is the alternative definitions of all statements whose operation depends on the type of output. The advantage of this approach is that in one file and in one place we have the possibility to compare definitions and change them at the same time.

3. Definition branching – the use of branching definition allows you to apply different options depending on the set option. What does \TeX branching look like? There are a number of options, we will choose the simplest of them:

You must first create a branch command. This is done with the command `\newif` for example, as follows:

```
\newif\ifpaper
```

This automatically creates two more commands: `\paperfalse` and `\papertrue`. The first one sets the `\ifpaper` condition invalid and the other valid. Then we can use the new branch command as follows:

```
\ifpaper \setmainfont{Constantia} \else \setmainfont{Tahoma}\fi
```

We then specify `\paperfalse` in the document as needed and from that point on, the parts in the branch between the word `\else` and `\fi` apply to the definitions. Conversely, by entering the command `\papertrue` we get the definitions for the valid condition `\ifpaper`.

In conclusion, we will just warn you that the concept of alternative definitions will be fully functional only if the document contains purely structural markup. So we add further justification that structural marks lead to efficient document creation.

National environment and special characters

Nowadays, in many documents it is necessary to use a number of characters that are somehow related to the set language. Word processing systems were mostly created in the environment of English-speaking countries, which use only 26 letters without accents and other peculiarities to express any text. It is very unique how systems based on the \TeX principle have been thoroughly dealt with typesetting in other languages since the early 1980s. The concepts created in these historical ages are no longer used to the same extent as before, yet it can be said that the tradition of respect for typesetting in other languages is maintained to this day.

The situation did not change much until the advent of universal character encoding with multibyte codes (UTF-8, UTF-16, UTF-32). The possibilities of various programs have reached the same level, and most of the previous solutions of various specialties could be simplified or abandoned altogether. Nevertheless, there are options that are given zero or minimal attention in other systems.

4.1 Input text encoding

As mentioned above, working with the locale requires efficient acquisition of the source text (more precisely, its content) and this is based on the ability to turn on the appropriate encoding. In the case of Czech, Slovak, Polish, Hungarian and similar geographically located texts, it is evident that coding is an indispensable prop. In all language areas, a staggering number of texts have already been produced in a variety of encodings. Although it is a modern effort to convert the necessary documents into one of the universal encodings, it is still necessary to work with the original settings. You can use the command in the preamble, for example:

```
\inputencoding{isolatin2}.
```

In the $X_{\text{Y}}\text{L}\text{A}\text{T}\text{E}\text{X}$ system, the default encoding is UTF-8, in which it is possible to use all the characters of the font used. In this context, however, it is necessary to emphasize that not every font is equipped with all the necessary characters.

4.2 Locale setting

In the system $X_{\text{Y}}\text{L}\text{A}\text{T}\text{E}\text{X}$ it is possible to set the locale by using the package `polyglossia`. There are also alternative methods (package `babel`, etc), but we will not deal with them for simplicity of interpretation.

By selecting the language, the corresponding hyphenation patterns are loaded, language-dependent strings are set (for example, “kapitola” in Czech instead of “chapter”) in English, some specific commands for using the font are selected, the date and time format is set, and so on.

We can use the already mentioned command to select the basic language

```
\setdefaultlanguage{lang},
```

whose parameter is the language name (a list of supported languages is available in the polyglossia package documentation). If you want to use more languages in the document, you can “preset” them with the command `\setotherlanguage{lang}`.

There are several ways to change the language in the text:

- The command

```
\text{lang}[options]{material}
```

will set the material in the set language.

- Environment with the name of the language, i.e. construction

```
\begin{lang}... \end{lang}.
```

- By the switching command

```
\selectlanguage{lang}.
```

4.3 Hyphenation algorithm and its settings

The \TeX -like systems use a hyphenation algorithm based on specially designed patterns, to which other less frequented or specific words can be added. Hyphenation patterns are created for many languages, and are set by selecting a language in the locale.

The hyphenation algorithm works with two values: the minimum number of characters to be left in the first part of the hyphen, and the minimum number of characters to be in the second part of the hyphen. The default values are 2, resp. 3. These parameters are set by commands `\lefthyphenmin = x` and `\righthyphenmin = y`. Sometimes it is appropriate slightly to loose these boundaries. For example, under predefined circumstances, the Czech word “osoba” (person) cannot be divided – but possible dividing points are “o-so-ba”. So two variants come into play: o-soba and oso-ba. However, none of them has at least two characters in the first part and at least three characters in the second part at the same time.

You can use the `\hyphenation{}` command to add exceptions or additional words to existing patterns used in the preamble, in the parameter of which whole words with indicated dividing points are written using hyphens. The separator between the words is a space, the number of words is not limited. Example:

```
\hyphenation{Post-script meta-lang-uage hardly}.
```

In the last word, we indicate the desire to completely exclude division.

If we do not want to create a global list of exceptions or release the character counts of individual parts of the hyphenated words, we can use “manual” hyphens in the appropriate place in the text. To indicate a hyphenation point, use the `\-` command, for

example `Post\script` or `meta\language` The division algorithm uses only these hyphenation points, no other, even if the word is in the exception list or hyphenation patterns can be applied to it.

4.4 Special characters and their solutions

Typographic principles and rules

The term **plain typesetting** means text typeset to one type, typeface and font size. This is usually the body text of the document. At first glance, it would seem that a plain typesetting is really nothing – just words arranged in sentences, paragraphs and pages. However, there are a number of characters in the common text that express various semantic and aesthetic differences in the typesetting and enable accurate and fast reading of the text. Some of these characters are precisely defined in the Czech spelling rules, some of which are part of the typographic rules. We will give at least a brief overview of these features for Czech language.

- Spaces – are the most common characters ever. We can divide them into inter-character and inter-word. Inter-character spaces are usually managed by the system itself (ligatures, kernings), in rare cases certain changes are made with them (for example, letter spacing or tracking – this variant of emphasizing is no longer used today).

Interword spaces can be normal, extended and narrowed. Normal interword spaces are used to justify the paragraph to a block, so they have a flexible size (correctly, its size should range from a quarter to a half of font size). Narrowed and widened gaps are solid.

- Hyphen – a horizontal bar connecting two words; used in accordance with the Czech Spelling Rules. There are never spaces around the hyphen. The hyphen that reaches the end of the line must be repeated at the beginning of the next line.
- Dash – semantically and graphically it differs from a hyphen. It has two meanings: it either replaces the punctuation mark, then spaces are inserted around it, or the second meaning when it replaces the words “to” or “versus”, then it is bet without surrounding spaces. In this second sense, he must not get to the line break – he must be replaced by the corresponding word. In American English there are two sizes of dash – the size of “punctuation” dash is equal to font size and the size of “interval” dash is a half of font size. The surrounding spaces are missing in both cases.
- Quotation marks – are twofold: initial and final. In common book fonts, the initial quotation marks are in the shape of nines and are placed on the baseline, the end quotation marks are in the shape of sixes, and are “hung” on the top line. In English there are different shapes: the initial quotation mark is in the shape of sixes and final in the shape of nines. Both of them are on the top line.
- Degree sign, percentage and per mille – these characters have a common feature in space: in Czech texts, if they are connected without a space to the previous number

(word), they become part of it and then form an adjective as a whole. Written with a space, they form two separate units with the previous word. These characters are typeset without space in English.

- The multiplication character \times is often incorrectly replaced by the letter “x”. It is added without a space if it forms a repetition expression (e.g. $5\times$), but with surrounding spaces it is typeset in the sense of a multiplication operation or expression of area dimensions, etc. (e.g. the surface is 10×15).
- Other characters – this includes ellipsis (three dots), the character et (&), asterisk, dagger, paragraph, etc. Usually these characters have a special space.

Technical solution concept

From a technical point of view, the way in which special characters are entered is solved, because most of them are not located directly on the computer keyboard. Therefore, there are different ways to express them, which can be used to insert the character into the computer and also to set the appropriate behavior.

Implementation in the \LaTeX system

The implementation of special characters is shown in the following table. It contains some commands, the definition of which is not implicit in the system, we have to do it ourselves. These are the following cases:

- Narrowed spaces – we need at least two: one quarter of font size for most cases and one sixth for space with ellipsis. The definitions are as follows:

```
\def\,{\penalty 10000\hskip 0.25em}
\def\;{\penalty 10000\hskip 0.16667em}
```

- Hyphen with automatic end-of-line detection and repetition on the next line:

```
\def\czhyph{\discretionary{-}{-}{-}}
```

- Dash with automatic end-of-line detection and possible replacement with the original word:

```
\def\to{\discretionary}{\hbox{to\ }}{--}}
```

Character	Source code	Example	Typeset output
Space	Spacebar press (even multiple)		
Narrow space	<code>\,</code>	<code>10\,mm</code> <code>J.\,F.\,Kennedy</code>	10 mm, J. F. Kennedy
Nonbreaking space	<code>~</code> (manually or by the program for automatic inserting after prepositions)	<code>a~thing</code> and <code>a~nothing</code>	a thing and a nothing

(to be continued on the next page)

Character	Source code	Example	Typeset output
Wide space	<code>\quad</code> (1 em) <code>\qquad</code> (2 em)	<code>1\quad Introduction</code>	1 Introduction
Paragraph	Empty line (even multiple)		
Three dots*	<code>...</code> or <code>\dots</code>	<code>silence\;\dots</code>	silence ...
Hyphen	- (directly from keyboard)	<code>on-line</code>	on-line
Hyphen	<code>\czhyph</code> (in Czech: this hyphen is doubled to the beginning of the next line during a line break)	<code>bude\spoj{}li</code>	bude- -li
Dash*	<code>--</code> (en-dash) <code>---</code> (em-dash)	<code>6--12</code>	6–12
Dash (to interval)	<code>\to</code> (this hyphen is replaced by the word “to” at the line break)	<code>6\to 12</code>	6–12, 6 to 12
Minus	<code>\$-\$</code> (in math mode)	<code>\$-10\$</code>	-10
Multiply sign*	<code>\$\times\$</code>	<code>\$2\times 3\$\,mm</code>	2 × 3 mm
Degree*	<code>^\circ\$</code>	<code>5\,^\circ\$C</code>	5 °C
Paragraph*	<code>\S</code> (only with number)	<code>\S\,36</code>	§ 36
Signs #, \$ a &	<code>\#, \\$, \&</code>		
Parenthesis { }	<code>\{, \}</code>		
Characters < >	<code>\$<\$, \$>\$</code>	<code>\$a>b\$</code>	$a > b$
Percent	<code>\%</code>	<code>10\%</code>	10%
Quotes*	<code>\uv{text}</code>	<code>\uv{thing}</code>	“thing”

* These characters can also be used directly from an UTF-8 source text.

For practice:

- Ex. 4.1: Typeset with special characters: We work from 8 to 14 hours. Even at temperatures below minus 15 Celsius degrees, the train in the Rocky Mountains overcomes an ascent of two and eight tenths of a percent. With a fifty percent probability, the bearded paragraph (in quotation marks) ten of the Study and Examination Regulations will be used twice.
- Ex. 4.2: Express with special characters that the mast is painted with yellow and green stripes, working hours are from 9 am to 5 pm, a match between Real Madrid and Barcelona is played on Saturday and that the lighting will cost about 150 US dollars.

Paragraph typesetting, algorithms, parameters

5.1 Typographic measuring systems, lengths, length registers

Before we will focused on to the paragraph typesetting, we need to turn a little and we will briefly deal with the measurement systems in typography.

For historical reasons, the field of typography and the production of printed materials uses special measuring systems developed around the 18th century. The units of measure are derived from the ancient measures (French foot, English foot) and were later slightly adjusted to the metric, resp. inch system.

The system used in continental Europe comes from France and was originally associated with the size of the French foot. In 1776, François Didôt published his manual for typewriters, where he determined the size of the basic typographic unit in relation to the metric measure. The base unit is **typographic point**; Didôt determined that 2660 points gives one meter, i.e. the size of the point is 0.376 mm. The larger unit is **cicero** = 12 Didôt points.

The system used in England and later widespread in America was based on the English foot, around the beginning of the 19th century, it was then connected to inch units in connection with the development of typesetting machines (monotypes). The basic typographic unit is also one **point** (marked 1 pt). Its size is equal to 1/72 inch, i.e. 0.353 mm. The larger unit is **pica** and is equal to 12 points.¹

It can be seen that both systems are similarly built, but there are differences between them. In addition, the development of computers has become a rather curious thing: software (as well as hardware) originating mostly from the USA was naturally equipped with an English measuring system. It has been adopted with this software in most European applications, so after several centuries, we also work with English points in continental Europe. This is good to know, because in some cases the typographic recommendations used in Europe relate to Didôt points, and for American software it is appropriate to convert to English points.

In addition to the mentioned absolute units, the units **relative** are also used. These are related to the size of the font used. There are two **em** (equal to the font size – approximately the width of the capital letter M) and **ex** (half of em, approximately the size of letter “x”). Relative dimensions are very often used to express dimensional relationships and their use is very efficient.

¹We will use the “pt” notation in connection with the T_EX system for the original English points of size 0.351 mm. Monotype points are also used in the T_EX system, but are called “bp” – acronym for “big point”. Also “pica” is the original English measure and is equal to 12 pt, not 12 bp.

5.1.1 Paragraph

Paragraph is considered a basic element of the document. Its shape and mutual visual separation have a fundamental influence on the overall appearance and effect of the document on the reader. The typesetting method is affected by several parameters, the appropriate values of which are given (see also Fig. 5.1).

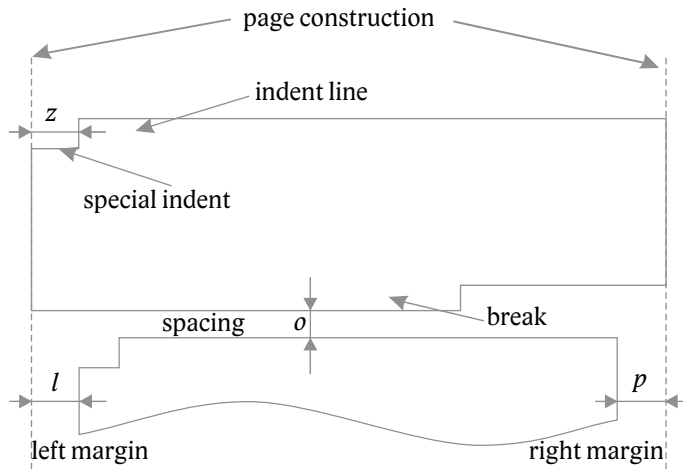


Figure 5.1: Parameters of paragraph typesetting

5.2 Body text typesetting parameters

Aligning – overall adjustment of individual lines of the paragraph. The basic option is **justify** (the left and right margins are aligned). Justifying is achieved by even changes in interword spaces and hyphenation.

Other alignment options are **ragged** (only the left margin is aligned, or only the right margin is aligned), or **centered**.

Line spacing – baseline distance of two consecutive lines. For optimal readability, it is set to about 120% of the font size. Its exact determination also depends on the font and the degree used.

Special indent – a space before the first character of the first line of a paragraph. Used for visual separation of paragraphs. Sometimes it is not used where visual separation already occurs – typically after the title. Its size varies between one and two em (for a ten-point font, i.e. 10–20 points), depending on the width of the lines.

Spacing – additional vertical space between paragraphs - creates another method of visual separation of paragraphs. Not used if special indents are used. Its size is typically half the line spacing. To achieve identical line spacing on all pages, the indent is set to a value equal to line spacing.

Left and right margin – additional space between the margin of the previous text and the margin of the paragraph on the left or on the right. Used for special paragraphs, such as enumerations (left margin), quotes (both margins), etc. The margin size corresponds to the size of the special indent.

Technical solution concept

The paragraph typesetting in computer programs is controlled by the line break algorithm. The quality of this algorithm determines the overall quality of the entire document. The basic goal of the right typesetting is to obtain as even a distribution of interword spaces as possible and an overall adjusted appearance. Line breaking algorithms can generally work with each line separately, or they work with the entire paragraph or other parameters. The problem of hyphenation and manipulation of some special characters is often included in the determining of line break.

Implementation in the X_YTeX system

A very sophisticated line breaking algorithm is implemented in TeX-based systems. In this respect, TeX does not have a more serious competitor among other computer typesetting programs.

To start a new paragraph, you need to insert a blank line (or more blank lines – they behave like one), or use the explicit command `\par`.

You can use the `\` command to move to a new line inside a paragraph. This command has an optional parameter that specifies the distance to the next line. Example: `\\[20mm]` – the command causes a vertical gap of 20 mm after moving to a new line. This command is never used as a new paragraph break.

Paragraphs automatically move to the next page when the page is filled. It is monitored that the page break does not occur in an inappropriate place in the paragraph, this monitoring can be affected by the following commands:

- `\widowpenalty=number` – an integer number in the range 0–10000 determines the “strength” of the penalty of the so-called widow – first paragraph line separately on the page end.
- `\clubpenalty=number` – an integer number in the range 0–10000 determines the “strength” of the penalty called orphan – last paragraph line separately on the new page start.
- `\brokenpenalty = number` – an integer number in the range 0–10000 specifies the “strength” of the page break penalty after the line that has a split word at the end.

You can use the `\newpage` commands to go to a new page (normal page break), command `\clearpage` (drop unplaced floating objects and go to a new page) or `\cleardoublepage` (dropping unplaced floating objects and moving to the next odd page).

By default, the typeset is provided with justifying, the special indent is equal to 15 points, line spacing is 12 points, spacing and both margins are zero. To change the default

typesetting, we need to set some length parameters, but we need to know something about the tool called **length register** and the possibilities of entering **length** in the \TeX system.

All dimensions are internally processed as integer values. Therefore, there are no rounding errors during handling. The unit is the so-called **scaled point** (sp), which represents a distance of $5.35 \cdot 10^{-9}$ m, i.e. about 5 nanometers. All other units represent only integer multiples of this dimension.

The \TeX system is unique in its ability to use different measurement systems. You can work with the English typographic system, the European typographic system, with inch or metric values. Usable units are:

<i>name</i>	<i>symbol</i>	<i>note</i>
historical english typographic point	pt	0.351 mm
monotype typographic point (big point)	bp	0.353 mm
pica	pc	1 pc = 12 pt
European Didôt's typographic point	dd	0.376 mm
cicero	cc	1 cc = 12 dd
inch	in	1 in = 25,4 mm
centimeter	cm	
milimeter	mm	
scaled point	sp	65 536 sp = 1 pt

The system also has relative units:

<i>name</i>	<i>symbol</i>	<i>note</i>
em	em	1 em = font size
ex	ex	1 ex = x-height

Flexible length is a length that, in addition to its natural size, has a defined stretchability and shrinkability interval. Flexible lengths are applied where the alignment algorithm works (for example, paragraph break, page fill). Maximal stretchability and shrinkability are defined by the keywords **plus** and **minus**.

Examples

Ex. 5.1: Flexible length 12 pt plus 1 pt minus 2pt can have a maximum size of 13 pt and a minimum of 10 pt.

Ex. 5.2: Length 2cm minus 0.35cm can be shrinkable only.

Ex. 5.3: Length 2in plus 35bp can be stretchable only.

Ex. 5.4: Special length `\fill` has zero natural size, but is infinitely stretchable.

Length registers

We can store the length in the **length register**. Some registres are already predefined, but you can also create your own. The registry has a name that has the form of a regular command. To insert a value into the register, use the command

```
\register = length
```

The length value setting applies within the group.

To use a register value, we write the name of the register, which can be preceded by a real coefficient by which the value of the register is multiplied. By multiplying the flexible length, it becomes an solid length equal to a multiple of the natural size (without shrinking and stretching).

Registers that relate to the paragraph typesetting:

`\textwidth` – width of typesetting (used in preamble),

`\textheight` – height of typesetting (used in preamble),

`\parindent` – special indent,

`\parskip` – spacing (flexible length),

`\baselineskip` – line spacing (cannot be set directly, it affects the font size setting or the `\baselinestretch` coefficient, which must be redefined),

`\leftskip` – left margin,

`\rightskip` – right margin.

Spaces

`\hspace` – horizontal space; in the parameter of this command there is any measure representing the required space. A space is created inside a line, not at the beginning or end. Example: `\hspace{2\parindent}`

`\hspace*` – a horizontal space command which works everywhere.

Example: `\hspace*{20pt}`

`\vspace` – vertical space; in the parameter of this command there is any measure representing the required space. A space is created inside the page, not at the beginning or end. The command can be used between paragraphs only, not within a paragraph. Example: `\vspace{5.5\baselineskip}`

`\vspace*` – a vertical space command which works everywhere.

Example: `\vspace*{\fill}`

`\smallskip` – command for a vertical space of 1/4 line spacing. Used only between paragraphs.

`\medskip` – command for a vertical space of 1/2 line spacing. Used only between paragraphs.

`\bigskip` – command for a vertical space of line spacing. Used only between paragraphs.

Examples

Ex. 5.5: `\textwidth=125mm` – setting the width of the text to 125 mm.

Ex. 5.6: `\parindent=2em` – setting the special indent to 2 em.

Ex. 5.7: `\footskip=5\baselineskip` – setting the page footer distance to five times the line spacing.

Ex. 5.8: `\parskip=20pt plus 3pt minus 2pt` – setting the flexible paragraph spacing.

Ex. 5.9: Affecting of line spacing: `\def\baselinestretch{1.5}\normalsize` – the line spacing increases to one and a half times the normal size. The command to set the font size causes the newly defined coefficient to be applied.

Ex. 5.10: `\leftskip=2\parindent \rightskip=\leftskip` – setting the margins of a narrowed typesetting (such as a quote) to two special indent size.

5.3 Other paragraph document elements

In addition to the body text, other parts in the form of paragraph material appear in the documents, but with other visual (and sometimes functional) parameters. These are various enumerations and lists, quotes, notes, captions of tables and figures, table fields, etc. For these paragraphs, we need to change the typesetting parameters appropriately to distinguish them from the body text and not leave the reader in doubt as to what element it is.

Implementation in the \LaTeX system

Paragraph environments

Several environments are used to change the implicit paragraph typesetting:

`center` – centering.

`flushleft` – left align (ragged right).

`flushright` – right align (ragged left).

`verse` – verse; the start of new verse is done by command `\`, at least one blank line is inserted between the stanzas.

`itemize` – items with starting character; each entry begins with a command `\item`.

`enumerate` – enumerated items; each entry begins with a command `\item`.

`description` – description; each entry begins with a command `\item[term]`.

`verbatim` – environment for literal text. A special environment in which no commands are interpreted and the typesetting exactly corresponds to the notation in the source text.

Examples

Ex. 5.11: Enumerated list:

Today's tasks:

```
\begin{enumerate}
\item Take the coat to the dry cleaners.
\item Buy 2 kg of potatoes.
\item Betting at the post office for a delivered package.
\end{enumerate}
```

Ex. 5.12: Description list:

What we learned at school yesterday:

```
\begin{description}
\item [National character encoding] -- there are several approaches
to display national characters on computer.
\item [Calculation of own information] -- one can find out how much
information it carries certain message.
\item [Data redundancy] -- is the difference between the maximum
possible entropy and the entropy of the given code.
\end{description}
```

For practice:

Ex. 5.1: By appropriately setting the typesetting parameters, type the selected text up to a width of 80 mm with special indents of one em, zero spacing with a stretch of 2 points, for one paragraph set the left margin to 1.5 times the special indent.

Ex. 5.2: Typeset two stanzas of the chosen poem.

Ex. 5.3: On a selected text consisting of at least three paragraphs, test the operation of the environment for centering and ragged left/right margins.

Ex. 5.4: Write a description of at least two components of a personal computer.

Mixed typesetting

A mixed typesetting uses different typefaces, different shapes and sizes. Since in most cases we will implement the entire document in one typeface, changes in shapes and sizes are most often needed. Changing the typeface can be accepted, for example, if we use a sans-serif variant of the same family. If we stick to props directly available in the \TeX distribution, an example might be Latin Modern Sans Serif combined with Latin Modern roman, but similar pairs can be seen elsewhere (Noto Sans + Noto Serif, etc.).

6.1 Use of additional typeface

In the X_{\LaTeX} system, the choice of font is solved in several possible ways. At this point, we already know that we choose the body font with the command `\setmainfont`, but we can also add an additional sans-serif and additional typewriter type to the document with the commands `\setsansfont{type}`, `\setmonofont{type}` respectively. We write these commands in the preamble. We can introduce any font at any time with the command

`\fontspec [options] {type}.`

Due to the X_{\LaTeX} technology, the name of any font that is installed in the given system can be written as type. This makes the possibilities of using fonts equal to practically any other word processing program.

6.2 Emphasizing

The most common reason for changing the section is the so-called **emphasizing**. This is an increase in emphasis on a certain section of the text. The basic way of highlighting is the use of *italics*. The italic font shape has sufficient ability to distinguish between the highlighted and ordinary part of the text during reading in detail, but does not create distinct areas when looking at the page as a whole.

Strong emphasis can be realized by **bold shape**. It attracts attention even with a quick glance at the entire page, so it is used when we consider this feature useful (for example, new terms in a textbook are an ideal case). Often, the bold face is also used for headings or other important and eye-catching elements of the page.

What if we want to emphasize something else in italic text? According to typographical rules, we will again use an plain shape.

If we want to emphasize something in bold text, we use *bold italics*.

We must realize two basic things in all emphasizing:

1. emphasizing shapes are always less legible than a normal shape, so we don't *waste* with emphasizing;

2. each emphasizing has its own function in the text, so we try to always implement the emphasize *consistently uniformly* so that the meaning of the shape changing is similar throughout the document.

An important rule is that if a part of text is marked with a certain shape, small punctuation is also provided with the same shape, if it belongs to this text, and the marked text in quotation marks or brackets also has the surrounding quotation marks and parentheses typeset with the same shape. If, on the other hand, an individual word is marked, followed by a comma or a period belonging to the entire sentence, the small punctuation is no longer emphasized.

Other available shapes are *slanted* and SMALL CAPS. The slanted shape has a weak distinguishing ability (compare with italics), so its use for emphasizing is not recommended, it can be used for change of graphic concept of the document. Small caps are often used for their dignified character to indicate names of persons, important titles, etc.

We DO NOT use underlines for emphasizing!

Underlining is a prop firmly connected to the typewriter and its possibilities, it has practically nothing to do in typesetting. A line in a composition has a completely different purpose – most often the separation of two units. In addition, underlining greatly reduces the readability of the text, as it collides with the descenders of the characters.

Nowadays we no longer use the so-called **expanded spacing** (increasing the spaces between characters) and certainly not the so-called **insertion**, i.e. the insertion of interword spaces between all characters (this is exclusively a typewriting prop). At the time of typesetting with metal letters, expanded spacing was used for economical reasons, so that a font of a different shape did not have to be produced. Today, this reason has completely disappeared, so the poor legibility of the expanded character spacing is not needed at all for normal emphasizing purposes.

Why do we change font size?

Appropriately used different font sizes significantly contribute to good orientation in the text. A logical and moderate choice of sizes in accordance with common rules therefore guides the reader and helps him in reading. In general, we decrease the importance with a smaller size, and with a greater size, we increase the importance of the text. We relate all font sizes to the main font size.

For example, what part of text can have a different size? Footnotes, image and table captions, tabular data, data in the running heads, etc. are typed with a smaller size, larger sizes are used mainly for headings.

We do not use font size change for emphasizing!

How much larger or smaller should the respective size be? The rule here is that a change of at least 20% is required for sufficient differentiation of the individual sizes. Therefore, it is not permissible for us, for example, to set the heading of one level in a 14-point font and the heading of a lower level in a 13-point font.

Implementation in the X_YL^AT_EX system

A change of shape or size is very often understood as a change of material in a certain part of a paragraph or a certain part of a document. This section can usually be defined

by a group (expressed by curly brackets or an environment). The commands for changing the font type, typeface or font size are therefore valid within the group, but some variants are also available that work with the material in their parameter.

Commands that affect the text to the end of the group can be applied to any length of text. Commands that work with text in a parameter can only be applied to material within a single paragraph.

Following commands are available for change of font type and typeface:

Font type and typeface changes		
<code>\textrm{...}</code>	or <code>{\rmfamily ...}</code>	<i>serif font type</i>
<code>\textsf{...}</code>	or <code>{\sffamily ...}</code>	<i>sanserif font type</i>
<code>\texttt{...}</code>	or <code>{\ttfamily ...}</code>	<i>typewriter font type</i>
<code>\textit{...}</code>	or <code>{\itshape ...}</code>	<i>italic shape</i>
<code>\textup{...}</code>	or <code>{\upshape ...}</code>	<i>up shape (from italic)</i>
<code>\textsc{...}</code>	or <code>{\scshape ...}</code>	<i>small caps</i>
<code>\textsl{...}</code>	or <code>{\slshape ...}</code>	<i>slanted</i>
<code>\textbf{...}</code>	or <code>{\bfseries ...}</code>	<i>boldface</i>
<code>\textmd{...}</code>	or <code>{\mdseries ...}</code>	<i>normal (from boldface)</i>
<code>\emph{...}</code>	or <code>{\em ...}</code>	<i>emphasizing</i>

The command `{\em ...}` is somewhat different. It behaves like this: If there is a normal typeface in its place, it turns on italics. If there is italic text in the place, it will turn on the upright shape. It precisely implements the typographical rule about nesting emphasizing, so it can be advantageously used for normal emphasize without having to “manually” check the correctness of the shape settings.

There are commands available to set the font size, the effect of which depends on the main font size setting. All commands have only one form that has an effect within its group. Automatically, with the setting of the appropriate font size, the correct line spacing in the paragraph will also be adjusted. In the overview, for basic orientation, we also list the point sizes that relate to the default setting of the main font to 10 points:

Font size changes	
<code>{\normalsize ...}</code>	<i>main font size 10 points (set by default)</i>
<code>{\large ...}</code>	<i>size 12,00 points</i>
<code>{\Large ...}</code>	<i>size 14,40 points</i>
<code>{\LARGE ...}</code>	<i>size 17,28 points</i>
<code>{\huge ...}</code>	<i>size 20,74 points</i>
<code>{\Huge ...}</code>	<i>size 24,88 points</i>
<code>{\small ...}</code>	<i>size 9,00 points</i>
<code>{\footnotesize ...}</code>	<i>size 8,00 points</i>
<code>{\scriptsize ...}</code>	<i>size 7,00 points</i>
<code>{\tiny ...}</code>	<i>size 5,00 points</i>

In addition to the listed commands that work with relative sizes, you can use the command `\fontsize{size}{line spacing}\selectfont` and thus enter any size and any line spacing.

For practice:

Ex. 6.1: Typeset: The circumference of the Earth was already calculated by ERATOSTHENES around 300 BC, when he used changes in the **observation angle** of the star SIRIUS on the same day *in two different geographical locations* – in ATHENS and in ALEXANDRIA. With the devices of the time, he got a *surprisingly accurate result*.

Ex. 6.2: Set the main font to Bookman and type the selected sentence in three sizes: 7, 10 and 17 points. Then do the same with the default Latin Modern font. In both cases, study the shapes of the letters in the lowercase and uppercase sizes. What conclusion can be drawn from this observation?

6.3 Technology – colors, models, definitions

One of the props that has become very widespread, especially with the possibility of electronic publishing and display on computer monitors, is **complementary color**. As its name already says, it is a complement to the basic color (black) and, according to typographical principles, it should create a sufficient contrast. Color shades are often used for this purpose, which are also economical in terms of price in printed publications, i.e. they are basic colors that do not require mixing and multiple passes through the printing machine. This is especially true for offset printing. In today's advanced printing machines, the source of which is a PDF file that is directly printed on paper without matrix preparation, the color requirements are different, but it is always advisable to think about possible applications where mixed color shades could create certain complications.

Working with colors in the X_YTEX system is enabled by the `xcolor` package, we will note the most basic options.

With the command `\color{color}` the coloring of the following printed material can be switched on. The set color is valid until the end of the group. The parameter is the name of the color, which is either predefined or can be created using the define command `\definecolor`.

An example of a color definition and a command for a new term in the textbook:

```
\definecolor{myblue}{rgb}{0.3,0.5,0.74}
\def\newterm#1{\color{myblue}#1}
```

Example of use (the text is then structurally marked):

The basic element of document is `\newterm{paragraph}`.

The command `\textcolor` works similarly with the difference that it sets the corresponding color for the material *in its parameter*. Example:

```
\textcolor{myblue}{Text printed with complementary color}
```

Another command is `\colorbox` – allows you to create a frame with a defined background color.

Example: `\colorbox{lightgray}{This is text on a gray background.}`

The result is: This is text on a gray background.

The frame can be bordered with another color, command: `\fcolorbox{a}{b}{t}`, example: `\fcolorbox{myblue}{lightgray}{Highlight text}` produces

Highlight text

The background color of the entire page can be set with the command `\pagecolor` – the background will start on the current page and continue until the next change or the end of the document. Example: `\pagecolor{lightgray}`

Document division

7.1 Heading systems

The text of a document consisting of a sequence of paragraphs is often divided into logical units that have their own headings. The correct system of headings significantly contributes to good orientation in the logic of the document, therefore its choice and implementation requires adequate attention.

Heading systems can look different, one of the most used is the *single different parameter* principle. That is, the headings have a common typeface and a common shape (for example, they are all bold), they differ only in size and the corresponding vertical spaces. The more important the heading (higher level), the greater the size. The second option is the same type and the same size, the difference is in the shape – for example, the most important heading is bold, the second level is italic, the third level is plain. The first mentioned system is more suitable for larger documents, the second is used in newspapers and magazines due to its space saving.

Especially in professional documents, headings can have numbering emphasizing individual levels. A hierarchical numbering (1 – 1.1 – 1.1.1) is often used.

Implementation in the \LaTeX system

The heading system is available with a set of commands: `\part`, `\chapter` (it is available in the document classes `book` and `report`), `\section`, `\subsection`, `\subsubsection`, `\paragraph`, `\subparagraph`. All of these commands have similar behavior and the same form – one optional and one mandatory parameter. The mandatory parameter is the own text of the title, an optional parameter can be another variant of the text that is used in the header and content. This set of commands does four things simultaneously:

1. types the heading text from the mandatory parameter in the appropriate font size and with appropriate vertical spacing, keeping the following paragraph on the same page;
2. the heading is provided with the corresponding number of the hierarchy;
3. creates an item of content; if an optional parameter is given, applies its text to the content;
4. places the text in the running heads (if used; for headings of the corresponding levels only), uses its material if an optional parameter is specified.

If we use any of the listed commands with an asterisk (ex. `\section*{...}`), then this command does only one thing – the typesetting of the corresponding heading (point 1 in previous list). The title is written without a number and no other actions are performed. This variant does not have an optional parameter. If we want to arrange any of the other three actions, we can use other commands and settings (see numbering systems, inserting material into running heads and transferring information to the table of contents).

7.2 Initials

A rather marginal prop used in special cases is the **initial** – a very noticeable highlighted and often decorative first letter of the following text. The initial can be conceived as just a very enlarged letter on the same line as the following text, the second option is a suspended initial, i.e. placed next to the paragraph on the left (with the line moved down so that its upper line aligns with the upper line of the first line), the third option is the so-called wrapped initial (wrapped around the text of the paragraph). In addition to the first option, this is technically a rather complex action that is not directly available with predefined commands, and for its programming it is necessary to know more about the technicalities of the rate. Given the scope of this material, we will not discuss initials further.

7.3 Table of contents creation

Automatically generated table of contents is commonplace in almost every word processing system these days. Again, on the one hand, the rules for how the content should look to be as useful as possible for readers can be discussed, and on the other hand, the technology that will implement the requirements in an efficient manner.

At least briefly on both aspects:

The decision of what should be and should not be in the content clearly belongs to the author's information. Of course, this is also related to the use of suitable structural markup. If the content is to contribute to the easiest possible orientation in the document, it must simultaneously fulfill two contradictory requirements: to have as little material as possible to be clear, at the same time to have as much material as possible to facilitate the understanding of the place to which it is referred. This, for example, is also related to the appropriate choice of heading text – too long headings are confusing, while too short headings do not provide sufficient detail about the following text.

The longer the document, the more challenging the design of the content shape. As one guide, the average difference between two consecutive pages of content items. At first glance, it can be assessed when the differences are too small, or even zero, that the content is too detailed, and vice versa. Sometimes it is also approached that the content is created in two versions: an overview and a detailed one. The reader then proceeds by finding an area of interest in the overview content and then finding the exact place they need to work on in the detailed content.

The content also includes lists of tables and figures, or other important document objects (graphs). The principles of their creation are similar, and it is also true that only the author of the document can responsibly assess their meaningfulness.

Implementation in the \LaTeX system

As previously mentioned, content items can be automatically created using heading commands. The place where the content should be placed is indicated by the command `\tableofcontents`. There are also predefined commands for the list of tables and figures: `\listoftables`, or `\listoffigures`.

Table and figure list entries are created by the `\caption` commands used for figure and table labels in `table` and `figure` environments.

Technically, the content is created in such a way that, during the compilation of the document, the necessary information is collected in an auxiliary file with the extension `.toc`, or `.lot` or `.lof`. In the second compilation, these files are read and the contents or appropriate lists are dumped from them. If the content is located at the beginning of the document and the second compilation and typesetting results in material is longer than one page, it is necessary to use another compilation to adjust the correct page numbers. So that we don't have to remember about it, it is convenient (at today's computer speeds and common document ranges) to immediately set up a threefold compilation of the document.

However, in the predefined variant, the headings of all levels are not inserted into the content. The level to be included in the content is stored in a special counter `tocdepth` and has a predefined value of 3. This means that the top three levels are transferred to the content (section, subsection, and subsubsection in the document class `article`). If we want to place, for example, only sections and subsections in the content, we simply write: `\setcounter{tocdepth}{2}` (see work with counters).

A content item can also be explicitly created other than by a command for some heading. The command `\addcontentsline` is available allowing to create an item corresponding to a heading, or the command `\addtocontents` to insert completely arbitrary material into the content. Examples:

`\addcontentsline{toc}{section}{content item}` – a table of content item is created as if it were the result of the `\section` command with relevant text.

`\addtocontents{lof}{\newpage}` – inserts in the content file for the list of figures (lof) a command to go to a new page (useful, for example, in a situation where we need to somewhat influence the default shape).

Examples

Ex. 7.1: Section level title with variant text in content:

```
\section[TTL integrated circuits]{Selected integrated
bipolar circuits}
```

Ex. 7.2: Unnumbered heading located in the document table of contents:

```
\subsection*{Notes on CMOS technology}
\addcontentsline{toc}{subsection}{Notes on CMOS technology}
```

Ex. 7.3: An unnumbered heading whose text is inserted into the running heads:

```
\section*{Attachments and comments}
\markboth{Attachments and comments}{Attachments and comments}
```

7.4 Technology – numbering, counters and references

In many cases in the document, it is necessary to use integer marks and associated functions (automatic value increase, listing). The so-called **counters** are used for this purpose. They are variables with which we can perform certain operations and refer to their values.

There are a number of ready-made, predefined counters. They are associated with commands for which they perform the necessary functions. For example, with the command `\section` a counter named `section` is associated and its role is to store the current section number. We can also create a counter ourselves, the definition command `\newcounter` is used for this. Examples:

```
\newcounter{ExampNum}  
\newcounter{RuleNum}[section]
```

In the second case, the definition command also allows creating a *dependency* between two counters – a new counter `RuleNum` is automatically reset when the parent counter `section` will increase its value by one.

After the definition of a new counter, the command `\thename` (for example `\theExampNum`) is automatically created, which is used to list (display) the value of the counter.

The following operations can be performed with each counter:

`\setcounter` – putting a value into the counter.

Example: `\setcounter{ExampNum}{10}`

`\addtocounter` – adding any value to the contents of the counter (both positive and negative). Example: `\addtocounter{ExampNum}{7}`

`\stepcounter` – increase the value by 1 and reset all dependent counters. Example: `\stepcounter{ExampNum}`

`\refstepcounter` – the same as `\stepcounter`, in addition, the value of label is set to the new value of the counter

`\value` – get the value of counter to use in arithmetic expressions

`\arabic` – counter value output method: Arabic numerals; similarly there is `\alph`, `\Alph`, `\roman`, `\Roman` and `\fnsymbol`, allowing the counter to be written as lowercase, uppercase, lowercase roman numerals, uppercase roman numerals, and footnote symbols respectively. Example:

`\def\theExampNum{\Alph{ExampNum}}` will cause the value of the counter to be printed in the form of capital letters A, B... For dependent counters, a more complicated form of listing can also be taken into account, for example:

`\def\theRuleNum{\thesection--\Roman{RuleNum}}` will write the counter `RuleNum` in the form 3–VII (if the `section` value is 3 and the `RuleNum` value is 7).

We will mention working with counters as needed for individual commands or situations.

There is one more useful function associated with the counters – the possibility of cross-reference, i.e. **reference**. The command `\label{label}` can be specified in the place (group, environment) where a counter is used, the value of which is adjusted with

`\refstepcounter` representing a variable into which the value of the counter is automatically filled. If we need to refer to this value, we use the command `\ref{label}` and the content of the corresponding label (i.e., the value of the counter) is transferred to the given place. Another option is to refer to the page number where the label is located with the command `\pageref{label}`. Example:

```
\label{here}We are in the section \ref{here} on page \pageref{here}
```

gives after processing: We are in section 7.4 on page 45.

Similar to the creation of content, cross-references are technically implemented by writing the necessary information to an auxiliary file – a file with the extension `.aux`. At least two compilations are required to get correct references.

Pages

8.1 Page elements

A page is usually divided into a typeface (the rectangle that contains the typed material) and margins (the free space on a sheet of paper). The typesetting pattern is further divided into a header, typesetting mirror and footer. The typesetting mirror contains the main material of the document, in the form of ordinary paragraphs of text, which can be supplemented with footnotes, marginalia, embedded images, tables and mathematical expressions. In the simplest case, the typeface consists of a sequence of smooth paragraphs, more complex is a multi-column arrangement.

8.2 Relationship between paragraph and page break

Paragraphs flowing in the type mirror can move to the next page according to certain rules:

- For each paragraph, there must be at least two of its lines on the same page. It is inadmissible for the first line to remain alone on the lower edge or the last line on the upper edge of the type mirror. (A technical solution to this problem has already been mentioned with paragraph typesetting.)
- Three lines of the following paragraph must remain on the same page after the heading. This requirement is partially solved – the heading and the following paragraph are automatically kept on the same page.
- Top-level headings (especially for longer texts) start on a new page (or on a new right – odd page). This can be solved by defining an appropriate command for the top-level heading that contains the `\clearpage` command or `\cleardoublepage` command.
- Some paragraphs (table fields, shorter quotations) cannot be divided by a page break at all. The solution is to use the so-called box, i.e. table field, frame command `\fbox` etc.

8.3 Page headers and footers

The page header is primarily used for better orientation in the document. In that case, it is a so-called **live header**, i.e. material whose content changes according to the context of the page content. Most often, the page number is inserted into the live header, followed

by the title of the chapter (section) in books, the name of the article and the name of the author in proceedings, the name of the section in magazines, etc. The page number is generally placed on the outer edge so that it is as easy as possible to reach when flipping through the document.

In the header, which has an optically very advantageous position, we use less noticeable typesetting elements – a smaller typeface, an additional shape. The header must be separated from the main text by a vertical space of at least one line of the main text, very often the separation is emphasized with a line.

The page foot has a similar role, but due to its optically disadvantageous position, it is usually only suitable for page numbering, for which a more conspicuous face (bold) or an enlarged size is sometimes used.

Implementation in the X_YLaTeX system

The headers and footers of the document are handled by special commands, which can be arbitrarily redefined as needed, or the smart package `fancyhdr` can be used. Due to the programmability of all components, the vibrancy of the header is very well ensured, and in this respect the X_YLaTeX type system is far ahead among typesetting systems.

Command `\pagestyle` – chooses how headers, footers and page numbers are displayed. It has one required parameter whose predefined values are: `empty` – no header, footer nor page number; `plain` – the page number is in the middle of the footer and the header is blank; `headings` – live headers are displayed with page numbers in the outer margin; `myheadings` – live header with slight user modification.

Command `\pagestyle` sets the page style from the position in source text onwards. If we want to change the page style for only one page, we can use a similar command `\thispagestyle` with a parameter of the same meaning.

With other registers, we can influence the dimensions of the header and footer:

- `\headheight` – a register setting the vertical size of the header;
- `\headsep` – a register setting header distance from text;
- `\footskip` – the register setting the distance footer baseline from the text.

After adding the package `fancyhdr` the `\pagestyle{fancy}` option is available and it is possible to make a number of settings allowing precise control of headers and footers. For more information, see the documentation for this package.

8.4 Footnotes

Footnote is a frequent element of professional documents in particular. It complements the main text with its content and allows the reader to study in more detail other aspects that are not necessary for basic reading due to fluency and maintaining attention. It is the author's responsibility to maintain an appropriate relationship between the content of the text and the content of the footnotes so that the stated objectives are met. The main text must remain completely understandable even if the reader does not process any footnote. However, sometimes we see a situation where the most important thing is

hidden in the notes; sometimes the notes represent a more significant part of the material than the main text. Both completely defeat the purpose of this prop.

In the main text there is a link to the footnote, on the same page at the bottom there is a separator line (it should be about 1/3 the width of the page width) and below it is the text of the footnote in a reduced size with a reference symbol. A reference to a note is always placed *without a preceding space* in the main text. It is also important what the link is linked to. If the link is meant to be a full sentence, the link is placed *after the punctuation*. If the reference is to a single word or phrase, it is placed *just after* that word or phrase.

Implementation in the $\text{X}_{\text{Y}}\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ system

A footnote is provided by the command `\footnote`, whose parameter is the text of the note. It automatically ensures that the note is on the same page as the text it belongs to. By default, a reference to a note is made up of a sequence number.

In some environments (tabular) or command parameters (headings), the footnote creation command is not functional and it is necessary to solve the whole situation in other (and sometimes quite complicated) ways. That's why there is another pair of commands allowing to divide the position of the link and the own text of the note into two actions – this is necessary, for example, in table fields. Where we need to have a link, mark with the command `\footnotemark` (is without parameters). After the incriminated environment, we will put the required note material with the command `\footnotetext` with the corresponding material in the parameter.

The counter `footnote`, is associated with footnotes. Its value and appearance can be influenced by the listed counter management commands (modify number value, modify numbering method). As a special mode of “numbering” can be used `fnsymbol` just for the purpose of formatting footnotes. This variant contains typical footnote reference symbols – asterisks, daggers, etc. instead of numbers.

8.5 Marginalia

Marginal notes, **marginalia**, are in some cases a very useful tool for orienting the reader in a document. Their material is very clear on the given page and can form clues that are very well searched for on the pages. For that reason, they are usually typed in a smaller typeface and a emphasized shape is used to distinguish them from the main text.

Implementation in the $\text{X}_{\text{Y}}\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ system

From a technical point of view, however, several settings need to be resolved: there must be enough space on the relevant side (for a double-sided document, it is always the outer side) and the marginal note is never divided by a page break. Both can be arranged with predefined commands. We will set the margins in the manner already mentioned, the following tools are then available to typeset the marginal notes:

- `\marginpar{...}` – a command for inserting a marginal note, for double-sided documents there is a possibility to distinguish the material of the left page and the

right page – the material of the left (even) page is written in the optional parameter before the mandatory parameter;

- `\marginwidth` – register setting marginalia width;
- `\marginparsep` – register setting the space between the text and the marginalia.

8.6 Page and paper

For printed documents, this is a relatively essential task. Improper placement of the typed material on a sheet of paper leads to the fact that the pages are overloaded, uneven, and the reader perceives the overall appearance of the document as unpleasant. If it is a larger document, we always design it as **two-sided**, we solve the parameters of the left (even) and right (odd) pages as well as the double page as a whole. Previously, due to technical reasons, some typewritten publications were created as one-sided (right pages only). Typewriting on ordinary paper sometimes completely defaced even the reverse side, so that no other text could be placed on it. In the era of high-quality laser printers, this technical problem is completely solved, and printing on only one side of the paper almost always just means wasting paper.

For electronic documents, the requirement for two-sided layout is somewhat reduced, but the work with blank margins remains very similar.

Placement of typeset material on a sheet of paper

Choosing the size of the edges is one of the classic tasks solved since the Middle Ages. Human perception of a page is not uniform – the page has its optical center of gravity elsewhere than the geometric center of gravity (approximately in the upper third). Also, the right part of the page appears heavier than the left, and it is advisable to lighten it somewhat by enlarging the right margin. Thus the margins can never be the same everywhere.

One of the simplest margin constructions used is the ratio of small integers. If we denote l the left margin, h the top margin, p the right margin, and d the bottom margin, then we express the margin ratio as follows:

$$l : h : p : d = 3 : 4 : 5 : 7$$

For a two-sided document, the left and right margins on the even (left) sides are swapped. If some space is needed for binding and trimming (see 12.3.2), it is subtracted from the row size of the paper and then the margins are determined.

Examples

Ex. 8.1: We want to set the margins and text width on B5 paper when a 25mm right margin is specified.

We determine the left margin from the margin ratio: $25 \cdot \frac{3}{5} = 15$ mm. From the size of the B5 format (176×250), we determine the text width: $176 - (25 + 15) = 136$ mm. Then the top and bottom margins are 20 mm and 35 mm, respectively.

Ex. 8.2: We have a sheet of A5 paper and want to type text width of 116 mm. What margins should we set?

The A5 format has dimensions of 148×210 mm. The sum of the sizes of the left and right margins is obtained by the difference $148 - 116 = 32$ mm. We have to divide 32 mm in the ratio 3 : 5, from here we get a left margin of size 12 mm and a right margin of size 20 mm. Furthermore, we can easily calculate the top margin (16 mm) and the bottom margin (28 mm) from the margin ratio.

8.7 Special pages design

Special pages mean, for example, the title page, page with publisher's record, imprint, various appendices, etc.

If we still follow to the principles of unity and contrast, we can also derive principles for the design of such pages.

The title page should use the same font as the main text (unity), if we use an additional font type for example for headings for clarity, we can also use it for inscriptions on the title page according to the same principle. In contrast to ordinary pages, however, the title page has a different design – larger font sizes, a central axis concept, the use of a complementary color, etc. We try to keep the amount of elements that we apply to such a page as small as possible (it is stated that we should suffice with three elements – e.g. three different degrees of font).

The publisher's record is usually typed to the footer, main font size, and plain typeface, individual information in paragraphs with ragged right. If there is a large amount of information, a font smaller than main size can be used.

We proceed similarly when editing other pages of this type (dedication, motto, imprint).

Implementation in the $X_{\text{L}}\text{A}_{\text{T}}\text{E}_{\text{X}}$ system

The layout of a page on a sheet of paper has a slightly different control in $\text{T}_{\text{E}}\text{X}$ type systems. While in other programs the margin sizes are determined directly, which then results in the text width, in the $X_{\text{L}}\text{A}_{\text{T}}\text{E}_{\text{X}}$ system, the text width is defined explicitly by the `\textwidth` register and the correct placement on a sheet of paper is achieved by shifting the implicitly given left and top text edge points to the required position using the length registers `\hoffset` and `\voffset`. By default, the upper left point of the text has a distance of 1 inch (25.4 mm) from the edge of the paper, but this is also affected by the settings of the program for processing and viewing the typeface. The text height, which is adjustable by the `\textheight` register, needs to be calculated from the required margins.

If we use running heads, it is necessary to calculate their size and the necessary space from the main text to the height of the page layout. The height of the page header is contained in the register `\headheight`, we have the space between the header and the main text in the register `\headsep`. It is somewhat different for the footer: in the register `\footskip` is the distance of the footer baseline from the main text, which is again a value that we add to the total height of the layout. Then the margins are calculated relative to this total height.

If we need to affect the margins in a different way on the odd and even pages of a two-sided document, there are two registers available: `\oddsidemargin` for odd pages and `\evensidemargin` for even pages. Their values are added to the registry setting `\hoffset`.

Many options for setting page parameters are provided by the `geometry` package, which for the sake of brevity we will not deal with in this text and refer the reader to its documentation.

Examples

Ex. 8.3: Set the necessary values to adjust the horizontal dimensions:

We want to typeset on A4 format, text width 162 mm with margins $l = 18$ mm, $p = 30$ mm. How do we set the appropriate registers?

Let's set the typeface width `\textwidth=162mm` and we process the document. By measuring from the result, we find that $l = 47$ mm. So we set `\hoffset=-29mm`.

Ex. 8.4: If we use the values of the previous example, we will also set the vertical dimensions accordingly. We need to have $h = 24$ mm, $d = 42$ mm, and this results in a text height of $297 - (24 + 42) = 231$ mm. We will set `\textheight=231mm`. After typesetting, we will find that the current top margin is 44 mm, so it will be `\voffset=-20mm`.

Ex. 8.5: Select a two-sided document on A5 paper size with a text width of 100 mm and set all other dimensions to process the typesetting.

Ex. 8.6: Type the title page of the document (center axe, main title to one third of the text height, large bold font, place information about the place and year at the bottom of the page).

Ex. 8.7: Test the section, subsection, and subsection headings in the document class `article`. Create the document table of contents on the selected material. Set running heads of type headings.

Mathematical and similar expressions

Mathematical typesetting has always been the biggest challenge for typesetters in history, and only the best were able to perform it well. This was also allegedly one of the main reasons why D. Knuth wanted to solve this problem with computer support. His system is quite clearly the best that can be used in computer typography for typesetting mathematics.

The typesetting of mathematical (as well as physical and chemical) expressions is subject to a large number of specific rules. The necessary unambiguity and precision of these expressions requires strict respect for all principles, because, unlike ordinary text, a fundamental shift in meaning can very often occur.

Each mathematical symbol has its unique and permanent shape (font, shape, size, surrounding spacing). It is irrelevant whether the symbol occurs in a separate expression or as part of the text of the paragraph. What should the symbols look like correctly? The answer to this key question is probably best solved by the ISO 80000 standard, which we recommend to consult if necessary. The set of mathematical and chemical expressions is elaborated very thoroughly in Nohel's somewhat older book (1976), from which a number of rules can be taken.

9.1 Elements of expressions

As building blocks from which expressions are composed, we recognize:

- Individual symbols. They can be variables (letters of the Latin or Greek alphabet) – they are always typed in math italics. Constants – numbers or named constants, such as the root of natural logarithms or Ludolf's number π , on the other hand, are always typeset with an upright (plain) typeface. In addition to constants, functions (sin, log), so-called address indices (F_{\max}), differential and some other objects are also typeset with a normal typeface.
- Operators. Plus, minus, etc. are symbols that also carry the corresponding spacing. For example, the minus symbol can act as a unary operator – denoting a negative or opposite value, in which case it is set without spaces – or as a binary operator – denoting a subtraction operation, then typeset with surrounding spaces.
- Indices and exponents. They are typeset with a reduced size, the exponent in the exponent then with an even further reduced size.
- Roots. The root has a horizontal line as long as the expression under the root and a height that exceeds the height of the rooted expression.

- Fractions and similar expressions. The fractional line of a fraction has fixed spaces from the expression in the numerator and in the denominator. The numerator and denominator are set to a slightly reduced size, and for compound fractions then to another reduced size. A similar element is the combination number.
- Sums, products, etc. Sums and similar operators have limit expressions below and above them, which can be set in the form of an index and an exponent (next to the summation sign) in case of a request for height reduction (in the text of a paragraph or in fractions).
- Large delimiters. These are large parentheses delimiting, for example, matrices or similar structures. Their size must be chosen to include the entire space of the embedded expression.
- Text. Accompanying words can be part of the expression (for example, “and thus”, “after substitution”). They are always typeset in the same way as the surrounding paragraph text. Sometimes verbal expressions are used instead of ordinary variables, for example, the formula for calculating profit can be written: profit = revenues – costs. These “word variables” are also always typeset with a normal shape.

Implementation in the \LaTeX system

The system can take care of most of the mentioned properties, but in some cases it is necessary to adjust the implicit typesetting method somewhat. The mathematics typesetting is concentrated in mathematical environments. You can use so-called **text math** inside a paragraph, and **display math** between paragraphs. The symbol used in text mathematics is adapted to the paragraph typesetting as much as possible, so as to disturb the line spacing as little as possible. It can therefore have a smaller size, other locations of limits can be used, etc. In display math, these restrictions do not apply.

There are four mathematical environments:

1. There are three options for text math: `$. . . $` or `\(. . . \)` or `\begin{math} . . . \end{math}`
2. There are again three options for display math: `$$. . . $$` or `\[. . . \]` or `\begin{displaymath} . . . \end{displaymath}`
3. For numbered display math `\begin{equation} . . . \end{equation}`

This environment automatically numbers the given expression, the number is written in parentheses on the right margin of the mirror. Example:

`\begin{equation} a+b=c \end{equation}` is typeset as

$$a + b = c \tag{9.1}$$

4. For numbered multiline display math `\begin{eqnarray} . . . \end{eqnarray}`

The environment allows you to align systems of expressions (for example, equations) and number each line (or omit the number). It is essentially a table that has three columns, the left column is automatically right-justified, the middle column is centered, and the right column is left-justified. Example:

```
\begin{eqnarray}
2y + 3x & = & 13 \\
5y - 10x & = & 15 \nonumber \\
y - 2x & = & 3 \\
\end{eqnarray}
```

yields

$$2y + 3x = 13 \tag{9.2}$$

$$5y - 10x = 15 \tag{9.3}$$

$$y - 2x = 3$$

Since this is a table and the “=” sign forms a separate column, there are spaces around it according to the settings in the `\arraycolsep` registry (see below); by default it is 6 pt. If we want to adjust the spaces, we set, for example, `\arraycolsep=2pt`, as recommended by one of the typographical rules, and we get:

$$2y + 3x = 13 \tag{9.4}$$

$$5y - 10x = 15 \tag{9.5}$$

$$y - 2x = 3$$

In any of the mentioned environments, mathematical elements and symbols can be applied in the same way with the following tools:

- Change typeface – math italics is on by default (italics without kerning); if necessary (constants, etc.), the typeface can be switched with the command `\mathrm`, `\mathit`, `\mathbf`, `\mathsf` (it is similar to working with typefaces in text, where the relevant commands have the prefix `\text` instead of `\math`).
- The comma – is the separator in a list of values with additional spaces. Hence the notation `1,2,3` gives 1, 2, 3. When writing a decimal number in Czech, however, we cannot tolerate a space after the decimal comma, so we cannot simply write, for example, `3,141`, which leads to 3, 141. It is necessary to write `3{,}141` (printed as 3,141) where the parentheses around the comma enclose the inner empty list and prevent unwanted whitespace.
- Exponent – is written with the character `^`, e.g. `c^2` yields c^2 or `x^{a+b}` yields x^{a+b} .
- Index – written with an underscore, e.g. `x_i` gives x_i or `a_{ij}` gives a_{ij} .

- Square root – command `\sqrt{}`, eg `\sqrt{a+b}` yields $\sqrt{a+b}$ or `\sqrt[3]{a^2+b^2}` produces $\sqrt[3]{a^2+b^2}$.
- Fraction – command `\frac{numerator}{denominator}`, eg `\frac{1}{2}` displays $\frac{1}{2}$ or `\frac{a+b}{c-d}` yields $\frac{a+b}{c-d}$.
- The number of symbols, Greek and other letters, functions – are available using the corresponding commands, an overview of which can be found in the literature (e.g. Rybička, 2003).

- Sums, integrals, limits... – symbols with typesetting limits above and below the symbol. They show the difference between text and display math.

Example: `\sum_{i=1}^{n-1} a_i` in text math yields $\sum_{i=1}^{n-1} a_i$ and in display math yields

$$\sum_{i=1}^{n-1} a_i$$

The symbol for the integral works similarly `\int`, product `\prod` etc. We can modify the implicit behavior with the command `\limits`, which always enforces the typesetting of limits above and below the sign, respectively, with the command `\nolimits`, which, on the other hand, forces the typesetting of limits next to the sign:

`\sum \limits_{i=1}^{n-1} a_i` inside the paragraph gives $\sum_{i=1}^{n-1} a_i$ and

`\sum \nolimits_{i=1}^{n-1} a_i` in display math gives

$$\sum_{i=1}^{n-1} a_i.$$

Limit `\lim_{x \rightarrow \infty} \frac{2\tau}{\tau-3}` is displayed as

$$\lim_{x \rightarrow \infty} \frac{2\tau}{\tau-3}.$$

- Large delimiters – parentheses, vertical lines, and some other marks sometimes need to be enlarged to properly surround the closed expression. Delimiters that have the ability to grow in this way are then set with a special pair of `\left` and `\right` commands. These commands must always be paired. If we don't want a delimiter on one side, we write it instead to the command `\left` or `\right` just a dot. For example, `\left\{\left[\frac{x^2}{\sqrt{x-2}}\right]\right\}` produces

$$\left\{ \left[\frac{x^2}{\sqrt{x-2}} \right] \right\}$$

- Text – we can use the command `\mbox{}`, the material in its parameter will be typed as an ordinary one-line text. Example:

`x + 2k^2 - y \leq 0 \quad \mbox{and from here} \quad y \geq x + 2k^2` gives

$$x + 2k^2 - y \leq 0 \quad \text{and from here} \quad y \geq x + 2k^2.$$

- Arrays and other structures – much like the environment available in text `tabular` (see later) for the typesetting of tables, `array` is available in the mathematical environment. All its options are identical to the `tabular` environment, including the way of writing parameters and individual table fields. One of the few minor differences is the register that controls the inter-column spacing is called `\arraycolsep` (in the `tabular` environment it is called `\tabcolsep`). For example

```
\left(
\begin{array}{r|ll}
a & b+c & 17 \\
b & a+c & 22 \\
c & a+b & 6
\end{array}
\right)
```

after processing yields

$$\left(\begin{array}{c|cc} a & b+c & 17 \\ b & a+c & 22 \\ \hline c & a+b & 6 \end{array} \right)$$

- Notation of matrices and vectors – according to the rules, variables of these types should be typeset with a bold shape. However, it is sometimes somewhat less successful graphically when we use the command for common font change `\mathbf`. There is an option to use the switch `\boldmath` used *outside* the math environment, which orders everything to be bolded. We can compare, for example, `\boldmath$a` (typeset as **a**) and `$_{\mathbf}{a}` (is output as **a**). The following definitions may be a useful aid:

```
\def\mbf#1{\mbox{\boldmath$#1$}}
\def\mbfs#1{\mbox{\scriptsize\boldmath$#1$}}
and their use:
```

In physics we can often meet with the expression

$$e^{-i\mathbf{k}t}$$

The angle α of two vectors \mathbf{u} and \mathbf{v} is defined by the following expression:

$$\alpha = \arccos \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \cdot \|\mathbf{v}\|}$$

After compiling we get:

In physics we can often meet with the expression $e^{-i\mathbf{k}t}$. The angle α of two vectors \mathbf{u} and \mathbf{v} is defined by the following expression:

$$\alpha = \arccos \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \cdot \|\mathbf{v}\|}$$

9.2 Inserting of expressions into the document, references

As already mentioned, mathematical expressions can be in paragraph text or in displayed form. The displayed form has a uniform shape: the expression is indented in the center and, if necessary, numbers are given on the right margin. The equation counter is pre-defined for expressions numbering and we can on the one hand influence the way it is written and on the other hand it can serve as a reference.

Example: We will use the command `\label` in the expression and we then refer to the defined label with the command `\ref` and `\pageref`

```
\begin{equation}
\mbox{Area} = \left[ \frac{f(A)+f(B)}{2} +
\sum_{i=1}^{N-1} f(x_i) \right] \cdot \Delta x
\label{trapezoids}
\end{equation}
Numeric integration of the function  $f$  on the interval
 $\langle A, B \rangle$  we can do by computing the expression
\ref{trapezoids} on the page \pageref{trapezoids}.
```

After typesetting we get:

$$\text{Area} = \left[\frac{f(A) + f(B)}{2} + \sum_{i=1}^{N-1} f(x_i) \right] \cdot \Delta x \quad (9.6)$$

Numeric integration of the function f on the interval $\langle A, B \rangle$ we can do by computing the expression 9.6 on the page 57.

For practice:

Ex. 9.1:

$$P = \sum_{i=1}^n g(A + ik) \quad (9.7)$$

Ex. 9.2:

$$y = \left[\frac{x(x^2 - 1)}{x + 1} - \frac{x + 1}{x(x - 1)^2} \right]^3$$

Ex. 9.3:

$$C \leq 2^N \quad (9.8)$$

$$\log C \leq N \log 2$$

$$N = \left\lceil \frac{\log C}{\log 2} \right\rceil \quad (9.9)$$

Ex. 9.4:

$$\text{success rate} = \frac{\text{number of correct answers}}{\text{number of examples entered}}$$

Ex. 9.5:

$$\lim_{\Delta x \rightarrow 0} \frac{f(x) - f(x + \Delta x)}{\Delta x}$$

Ex. 9.6:

$$\text{discontinuous function } g(\xi) = \begin{cases} 0 & \text{pro } \xi < 0 \\ 2\xi & \text{pro } \xi \in (0, 1) \\ 2 & \text{pro } \xi \leq 1 \end{cases}$$

Ex. 9.7:

$$\text{transfer matrix: } \begin{bmatrix} \alpha_{11} & \beta_{12} \\ \gamma_{21} & \delta_{22} \end{bmatrix} \quad (9.10)$$

Tables

Tabular and sequential typesetting solves the problem of vertical alignment of document elements. Since, in contrast to typewriting, the width of each character is different in typesetting, the rather simplified principle cannot be used – inserting *strokes* to achieve the intended dimensional effect. While a typewriter stroke is a unit of length that was normalized to 1/10" on typewriters and could be used to measure any text, in typesetting we have to use completely different means. They are usually quite tightly connected to the technology used and we can divide them into two groups: **tab stops** and **tables**.

10.1 Table types

There are **open** tables, when the data is placed in columns usually completely without vertical delimiting lines, the header row of the table and the bottom (total) row are then separated by horizontal lines. Tables of this type are suitable for simpler cases and smaller amounts of data. In some cases, tabulation stops can be used for their implementation.

So called **closed** tables have individual fields delimited by lines, which can be double-thick to indicate a more complex structure (blocks of columns).

Fields or larger units in a table can be highlighted with a background color, with or without border lines.

10.2 Ways to align tabular data

The tabbing environment

It is about implementing the principle of tab stops. The most important elements are the statement to set the tab stop and the statement to use the position of the tab stop. The space between tabulation stops automatically forms a separate group, all possible settings of typefaces, etc. apply only to the extent of one field.

Example

tabbing environment example

```
\begin{tabbing}
\bfseries City \hspace{30mm} \= % setting tab stop
\bfseries Temperature at 7 AM \|[5pt]
Philadelphia \> 17.5$^\circ$ C \ \ % the use of tab stop
New York \> 16.8$^\circ$ C \ \
Washington \> 15.9\=$^\circ$ C \ \ % setting second tab stop
\> \> uncalibrated scale \ \
Boston \> 19.5$^\circ$ C
\end{tabbing}
```

After typesetting we get:

City	Temperature at 7 AM
Philadelphia	17.5° C
New York	16.8° C
Washington	15.9° C
	uncalibrated scale
Boston	19.5° C

The tabular environment

Tables in this environment have various interesting possibilities. Compared to interactive programs, working with tables is quite different, some cases are quite difficult, but on the other hand, there are a number of tools that allow you to automate typesetting, and thus significantly simplify work for common cases.

We will show a few examples from the wide repertoire of possibilities.

Basic table form

```
\begin{tabular}[vertical position]{column definition} \hline
field & field & ... \\ \hline
...
field & field & ... \\ \hline
\end{tabular}
```

vertical position – indicates the position relative to the surroundings. Implicitly, the table is placed vertically in the center to the surroundings, by entering this parameter in the form of the letter *b* the table is placed at the bottom edge, with the letter *t* upper edge relative to the surroundings.

column definition – a sequence of characters symbolically representing table columns, their alignment, and the material of the inter-column space. Letters *l*, *c*, *r* means left, center, and right aligned column respectively. Notation *p{measure}* means a justified column of width *measure*. The vertical line represents the inter-column space filled with a vertical line. Command *@{...}* allows you to define the material of the inter-column space, which will automatically be repeated in each row.

field – a table field is a separate group with any one-row material.

Simple table example

```
\begin{tabular}{rc}
\bfseries Points & \bfseries Grade \\ \hline
$<$50 & insufficient \\
50--69 & good \\
70--89 & very good \\
90--100 & excellent
\end{tabular}
```

After processing we get:

Points	Grade
<50	insufficient
50–69	good
70–89	very good
90–100	excellent

In the following table, we show the two necessary props – the merging of the table fields and the two alignment methods.

If we need span a table field over several others, we use the command

```
\multicolumn{number}{alignment}{text}.
```

This command allows you to define a different way of alignment and inter-column materials in the newly created merged field. Therefore, it can also be used to simply re-define the alignment method and inter-column material of a given field without merging multiple fields.

The correct vertical alignment of the material of the table fields is a very essential element, especially for numerical values. The numerical values must be aligned so that the same decimals are below each other. This is not completely trivial for values with different numbers of digits.

Two alignment methods are shown in the example. One consists of splitting into two columns (shown on the “score” column). The second method shown in the “points” column pads each value with an extra space of the same width, the column can then be centered. The command `\hphantom{0}` provides a space that is exactly equal to the width of the digit and for better readability and usability, we defined this space with a shorter command `\fm`.

Table with border

```
\def\fm{\hphantom{0}}
\begin{tabular}{|r||l|r@{:}r|c|} \hline
\bfseries 0. & \bfseries Country & & & \\
\multicolumn{2}{|c|}{\bfseries Score} & & & \\
\bfseries Points & \\\hline\hline
1. & Italy & 23 & 4 & 14\\\hline
2. & France & 18 & 7 & 10\\\hline
3. & Sweden & 14 & 10 & \fm 8\\\hline
4. & Switzerland & 3 & 18 & \fm 3\\\hline
\end{tabular}
```

After typesetting we get:

O.	Country	Score	Points
1.	Italy	23: 4	14
2.	France	18: 7	10
3.	Sweden	14: 10	8
4.	Switzerland	3: 18	3

Influencing the row spacing in the table can be done by redefining the `\arraystretch` coefficient, for example `\def\arraystretch{1.4}`. The size of the inter-column space can be influenced by the value of the registry `\tabcolsep`, whose default value is 6 pt.

We modified the previous table by setting `\def\arraystretch{1.3}` `\tabcolsep=15pt` and we get:

O.	Country	Score	Points
1.	Italy	23: 4	14
2.	France	18: 7	10
3.	Sweden	14: 10	8
4.	Switzerland	3: 18	3

To practice the table typesetting, we recommend do following tasks:

For practice:

Ex. 10.1: Using the `tabbing` environment typeset:

```

procedure Number(var C: longint);
var R: string;
    Position: byte;
begin Position:=5;
        while not eof do begin
            readln(R);
            writeln(C: Position, '-', R)
        end
end;

```

Ex. 10.2: Typeset a small multiplication table.

Ex. 10.3: Typeset your personal timetable.

10.3 Inserting tables into document

Tables are an essential element in professional documents and need additional props for their inclusion. Each table should have its own label, and inserting a table greatly reduces the possibility of page breaks. Both problems are solved by the `table` environment and inside it the command `\caption`.

Environment `table` is a so-called **floating environment**. It means that the resulting document may have *in a different place* than in the source text. The page build algorithm makes it possible to detect that the content of the environment no longer fits on the current page, store the material of the entire environment in memory and dump it only after switching to a new page (or even later). We will avoid all kinds of empty spaces resulting from missing tables in a given place.

In most cases, the floating environment has a positive effect on the efficiency of creating a precise document. However, there are cases when the general algorithm is not suitable for solving a more complicated situation, then there may be a variant that the floating environment cannot be placed sensibly and all the tables are “summarized” at the end of the chapter or the end of the document. Then comes the somewhat more demanding “manual” work of adjusting the predefined parameters of the placement algorithm, or even leaving floating environments and typesetting in another way.

Example: Optional parameter of `table` environment is a sequence of letters helping the embedding algorithm. The order of the letters determines the desired priorities. The letter `h` says that the first thing to try is to place the table exactly where it is in the source text. If it doesn't fit, then the `t` instruction starts, which is the placement on the top of this or next page. Analogously `b` is this or next page bottom part and finally `p` is placed on a completely separate page. We will always use this specification if it is a larger table, for which we know in advance that it will not fit in common places.

Floating environment example

```

\begin{table}[htbp]
\caption{Group C standings after ten contestes played}

\begin{tabular}{|r||l|r@{:}r|c|} \hline
\bfseries O. & \bfseries Country & 
\multicolumn{2}{c|}{\bfseries Score} & 
\bfseries Points \\ \hline\hline
1. & Italy & 23 & 4 & 14 \\ \hline
2. & France & 18 & 7 & 10 \\ \hline
3. & Sweden & 14 & 10 & \fm 8 \\ \hline
4. & Switzerland & 3 & 18 & \fm 3 \\ \hline
\end{tabular}
\end{table}

```

Tab. 10.1: *Group C standings after ten contestes played*

O.	Country	Score	Points
1.	Italy	23: 4	14
2.	France	18: 7	10
3.	Sweden	14: 10	8
4.	Switzerland	3: 18	3

With the `table` environment a counter of the same name indicating the current table number is connected. Its value is updated with the command `\caption`. If it is not specified for a table, the table is not counted. By placing the command `\label{...}` inside the command `\caption{...}` or behind it within the `table` environment creates a link that can be used with `\ref{...}` commands or `\pageref{...}` similarly to the case of equations. The table caption is also automatically inserted into the content file with the extension `.lot` and allows creating a list of tables with the command `\listoftables`.

Image material and graphics

In professional documents (and not only in them) the presence of various graphic elements is absolutely indispensable. These are often various schemes, drawings, graphs, diagrams, but also photos, drawings, illustrations or elements affecting the aesthetic level of the entire document – ornaments, lines, etc.

11.1 Types of images and their sources

A text processing system such as $\text{X}_{\text{T}}\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ is not primarily intended for the creation and processing of image information. It is therefore quite natural that the image elements come from some other software – a graphic editor capable of any modifications. The resulting edited image is then transferred to the document in the form of a computer file of a precisely specified graphic format and incorporated in a suitable place. At this point, some simple adjustments can still be made, such as resizing (detailed adaptation to the given location) or clipping.

Computer graphics are generally available in two different qualitatively quite different forms – raster and vector. Without going into details, let's just remind you that raster graphics as a collection of elementary points (pixels) are used mainly for photos and drawings, their quality is fundamentally dependent on the character of the output device (especially on the possibilities of displaying a certain density) and it has great limitations in terms of editing and modification options. On the other hand, vector graphics defined as a sequence of image objects can always be displayed in maximum quality and also provides a wider repertoire of editing actions that do not affect the output quality, which is, of course, balanced by far more complicated processing procedures.

It directly follows that different technical forms are also suitable for different types of visual information, and it is a big mistake to use raster technology for diagrams and graphs, for example. If we are working with a system whose principles include a significant effort for quality output, it is necessary to project this effort into the sources from which we take image information.

The insertion of graphic information into the document is made possible by the fact that the output format of the document (PDF) is composed of the typesetting solved using the $\text{T}_{\text{E}}\text{X}$ u tools and the content of the inserted graphic file. At the time of typesetting, the $\text{T}_{\text{E}}\text{X}$ translator (and all extensions) has no information about what is in the image file, it just inserts the rectangle as if it were a letter. So the only information lies in the dimensions of this rectangle. Only after the entire typesetting is done, the real content of the inserted files is inserted into the prepared rectangles and the result is converted to PDF format.

In earlier times, when the output format after translation by the $\text{T}_{\text{E}}\text{X}$ translator was by default DVI (DeVice Independent), graphics were solved only with tools equivalent to

working with fonts. Certain remnants of these technologies can still be used today, but they do not constitute the majority way of working with graphics as it was in the past.

With the development of output to PostScript and later PDF formats, the palette of options for working with graphics also expanded to include the ability to control the output driver using commands in the source text, i.e. control the PostScript language interpreter or the PDF format viewer. Internally, this is arranged by the command `\special`, which the compiler just forwards to the output, and there it is processed by the display program.

Very extensive packages with almost unlimited possibilities for creating graphic elements are based on this principle: PSTricks (to PostScript output format), PDFTricks (similar to PDF output format), TikZ (PDF output) and the METAPOST specialty that additionally uses Knuth's language for defining METAFONT fonts. The descriptions and capabilities of the mentioned tools go far beyond the scope of this text, so we will not discuss them further.

In the field of raster graphics, there are a plethora of file formats available, each with certain specific characteristics. JPEG and PNG raster files can be inserted into a $\text{X}_{\text{T}}\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ document, which are widely available and widely used alternatives to which a variety of other formats can be converted in image editors. It can be said that these two formats practically cover the entire area of all raster information.

Vector formats are represented by two variants – the Adobe PostScript format (including its single-page Encapsulated PostScript form) and the PDF format. Since Adobe PDF is a kind of ideological clone (perhaps even a successor) of the more general concept of the PostScript format, it is natural that it is a completely native part of the output form of the document, which is also produced by default in the PDF format.

Implementation in the $\text{X}_{\text{T}}\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ system

The `graphicx` package is used to work with image information (its older form `graphics` can also be used, which provides a completely different way of working with optional parameters).

The command

```
\includegraphics[options]{file}
```

provides the insertion of an image file into the document. In its basic form, the specified file is inserted into the given location of the document; a rectangular space is created, the dimensions of which are read from the graphic file (bounding box). The bottom edge of the rectangle is placed on the line of the current typesetting point.

An optional parameter can include a sequence of specifications in the form *key = value*; these specifications are separated by commas. We will list some of them for illustration – more detailed information can be found in the complete documentation for the given package.

A big advantage over the usual approach of other systems is the possibility to set dimensional adjustments according to the values of own and predefined length registers. For example, an image can always be automatically adjusted to the width of a column, its height can be 2/3 of a page, etc. For inserting images and editing graphic elements, you can create corresponding custom commands, which then ensure the complete identity of the visual form throughout the document and significantly facilitate typesetting and

subsequent editing. In case of any need for overall modification of the document (format change), all objects are automatically adapted to the new situation.

Selected options for editing the embedded image in the optional parameter:

width – allows you to set the width of the inserted rectangle (transforms embedded file). The height of the rectangle changes proportionally, unless otherwise specified.

height – allows you to set the height of the inserted rectangle.

viewport – a part of the image specified by relative coordinates in the image rectangle. It is written as four values: the coordinates of the lower left corner of the viewport and the coordinates of the upper right corner. This will affect the size at which the rate will be calculated, but the image will be displayed in its entirety. Therefore, if the viewport is smaller than the original rectangle, some parts of the image may overlap the surrounding material.

trim – similar to viewport, but the four values specify how many points are cut (and added when negative) from the left, bottom, right, and top sides of the rectangle.

clip – crops the view to the set rectangle. Suitable in combination with viewport or trim.

angle – rotates the image rectangle (including content) by the appropriate angle counterclockwise.

origin – center point of rotation. It is entered as a letter (or a combination thereof) specifying one of the reference points: l (left), r (right), b (bottom), t (top), c (center).

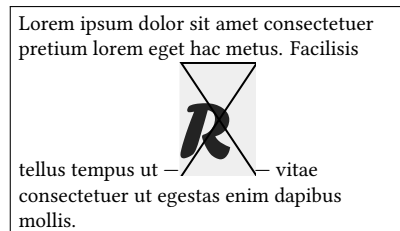
scale – rescaling the image. The magnification/reduction coefficient is entered, which is used to multiply the original dimensions of the image.

draft – does not insert the file, but only displays the rectangle in which the graphic should be placed. It significantly reduces the size of the output file and speeds up translation.

Examples: Suppose we have an image in the file `a.pdf` that has a rectangle size of 10×15 mm.

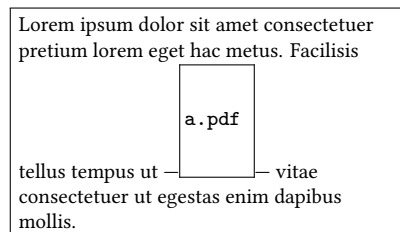
• insertion without transformations:

```
\includegraphics {a.pdf}
```



• embed without graphic data (suitable for debugging):

```
\includegraphics[draft] {a.pdf}
```



- inset with a change in width, the height will be adjusted proportionally to the original dimensions of the image:

```
\includegraphics [width=7mm]
{a.pdf}
```

- inset with a change in height, the width will be adjusted proportionally to the original dimensions of the image:

```
\includegraphics [height=10mm]
{a.pdf}
```

- insertion with change of width and height:

```
\includegraphics [width=15mm,
height=10mm] {a.pdf}
```

- viewport (i.e. mutual displacement of the embedded graphic and the rectangle):

```
\includegraphics [viewport=10 10 50
60] {a.pdf}
```

- viewport and crop the view to new dimensions:

```
\includegraphics [viewport=10 10 50
60, clip] {a.pdf}
```

- crop, a negative value increases the dimension:

```
\includegraphics [trim=-10 10 20
15] {a.pdf}
```

- crop and display to new dimensions:

```
\includegraphics [trim=-10 10 20
15, clip] {a.pdf}
```

Lorem ipsum dolor sit amet consectetur
pretium lorem eget hac metus. Facilisis



tellus tempus ut — vitae
consectetur ut egestas enim dapibus
mollis.

Lorem ipsum dolor sit amet consectetur
pretium lorem eget hac metus. Facilisis



tellus tempus ut — vitae
consectetur ut egestas enim dapibus
mollis.

Lorem ipsum dolor sit amet consectetur
pretium lorem eget hac metus. Facilisis



tellus tempus ut — vitae
consectetur ut egestas enim dapibus
mollis.

Lorem ipsum dolor sit amet consectetur
pretium lorem eget hac metus. Facilisis



tellus tempus ut — vitae
consectetur ut egestas enim dapibus
mollis.

Lorem ipsum dolor sit amet consectetur
pretium lorem eget hac metus. Facilisis



tellus tempus ut — vitae
consectetur ut egestas enim dapibus
mollis.

Lorem ipsum dolor sit amet consectetur
pretium lorem eget hac metus. Facilisis



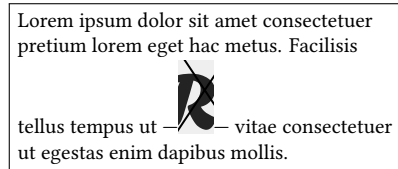
tellus tempus ut — vitae
consectetur ut egestas enim dapibus
mollis.

Lorem ipsum dolor sit amet consectetur
pretium lorem eget hac metus. Facilisis

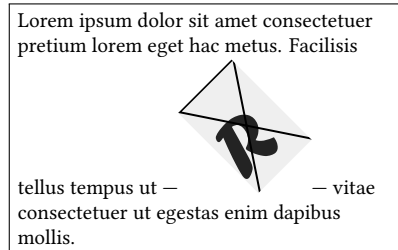


tellus tempus ut — vitae
consectetur ut egestas enim dapibus
mollis.

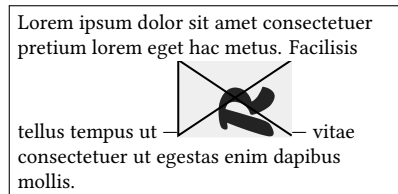
- normal trim – positive values:
`\includegraphics[trim=5 5 10 10,
clip] {a.pdf}`



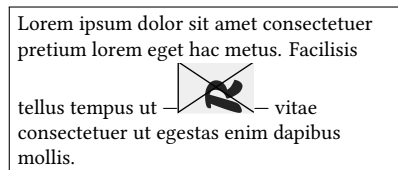
- image rotation by 45°:
`\includegraphics[angle=45] {a.pdf}`



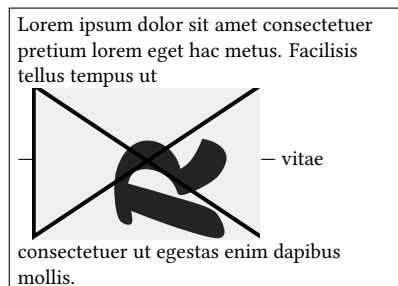
- rotation and adjustment of the resulting height:
`\includegraphics[angle=90,
height=10mm] {a.pdf}`



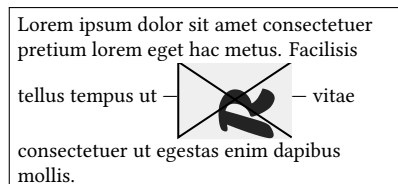
- height adjustment and subsequent rotation; here and in the previous example it can be seen that the parameters are processed and applied in the order in which they were written:
`\includegraphics[height=10mm,
angle=90] {a.pdf}`



- rotation according to the center point on the lower edge of the image, adjusting the height above the line to 10 mm:
`\includegraphics[origin=bc,
angle=90, height=10mm] {a.pdf}`



- rotation as in the previous example, but the height adjustment applies to the total size of the image, not just the part that is above the baseline:
`\includegraphics[origin=bc,
angle=90, totalheight=10mm]
{a.pdf}`



11.2 Graphic elements in the document

As the graphic elements we can suppose a wide range of different props, which can be divided into several groups. We especially notice elements inserted from special fonts and elements created by graphic operations.

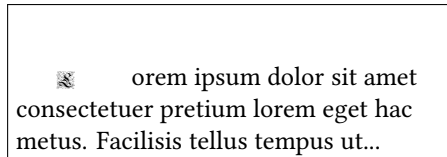
11.2.1 Special fonts characters

Ornaments formed by special characters of a certain font are one of the simplest graphic props. These include, for example, visual separators of logical units (end of chapter mark, etc.), decorative initials, warning signs, link symbols, etc. Since this is the same operation as inserting ordinary text, these elements do not require any special tools, work in any situation and in each compiler configuration and output format.

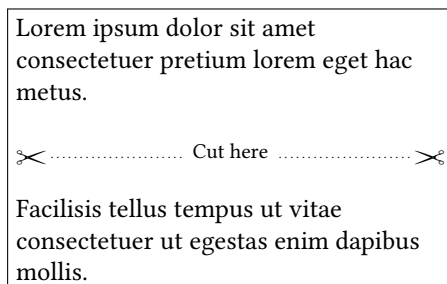
Characters can be colored in cooperation with the color system (already mentioned), which opens up more possibilities for their use.

Examples:

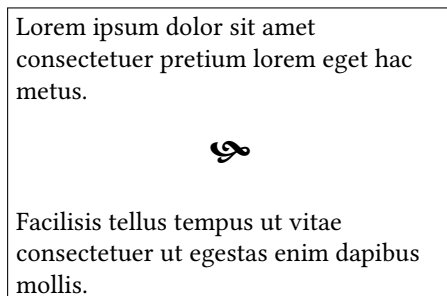
- Decorative initial, font yinit.otf (author Yannis Haralambous):



- Scissors symbol, free font Ozdoby.otf:



- Decorative logical unit separator (character from the wingdings font):



11.2.2 Typesetting elements and operations with them

In this section, we will deal with lines, rectangular (colored) areas, various frames and their transformations and arrangements.

For this we will need at least a cursory look into the “kitchen” of the typesetting machine – we will look at the basic principle of how this machine works and what elementary operations it has.

11.2.3 Typesetting principle and T_EX modes

In principle, the method of typesetting is very simple: rectangular frames (**boxes**) representing individual letters are gradually folded one after the other, possible spaces are inserted between them – empty spaces (**glue**) and even a third element – the so-called **penalty**, i.e. some kind of auxiliary information influencing the decision on the further progress of the rate.

At the time of typesetting, the typesetting machine doesn't really care what the graphic look of the move hidden inside the box is, it only cares about the geometry – the reference point, width, height and depth of the frame. The reference point is implicitly placed on the baseline just behind the previous box. A description of the box and an example of the typesetting of one word as seen by the typesetting machine is shown in Fig. 11.1.

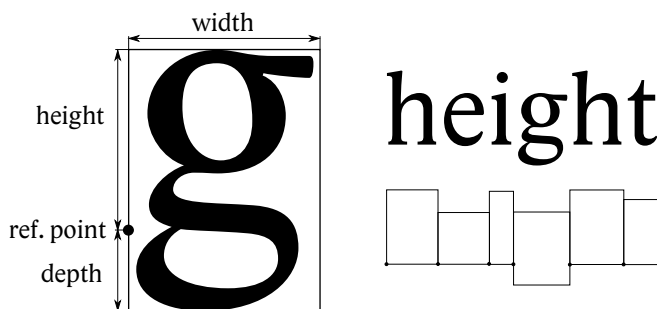


Figure 11.1: Box description and basic typesetting method

In addition to the boxes formed by the individual characters of the given font, there are other boxes, and we can also create them. A box is a typeset line after the completion of a line break in a paragraph, a box is a typeset paragraph and a typeset page. A box is also a table or embedded image. In addition, we can create our own boxes:

- LR (left-to-right) box – created with the command `\makebox` or its simplified version `\mbox`,
- paragraph box – created with the command `\parbox` or with environment `minipage`,
- rule box – created with the command `\rule`.

The T_EX system works with typesetting boxes in a six different modes. Now it will be clearer how the typesetting machine behaves and why some commands work in some places and not in others:

1. Main vertical mode – the situation when the typesetting is *between paragraphs* and vertical page alignment and page breaks are handled here, among other things. For example, `\vspace` or `\bigskip` commands work here.
2. The main horizontal mode – the typesetting is located inside the paragraph and, in addition to its own lines, possible alignment (`justify`, `center`, `right`) and line breaks are solved here. For example, `\hspace` or `\dotfill` commands work here.

3. Restricted horizontal mode – work in a single-line box (for example, a table field). This is the base typesetting without handling alignment and line breaks.
4. Restricted vertical mode – work in the so-called vertical box, where there is no page break (table, floating environment).
5. Text math mode (see description of math environments).
6. Display math mode (see description of math environments).

Sometimes it is important to know how to switch between individual modes, or to force this switch with the corresponding command.

The switch from the main vertical to the main horizontal mode is done by writing some character (this is assumed to be the beginning of a new paragraph) or by the commands `\noindent` – starts a new paragraph and at the same time cancels any paragraph special indent; `\indent` – starts a new paragraph including special indent; `\leavevmode` – exits vertical mode but takes no further action.

The easiest way to switch back from the main horizontal mode to the main vertical mode is with an empty line (ends the paragraph), or with the command `\par`. Some environments do the same (e.g. `center`).

Restricted modes arise in boxes. Transitions to and from mathematical modes ensure the boundaries of all mathematical environments, as already mentioned in the relevant chapter.

11.2.4 Boxes

In the following overview, we will present the options for creating your own boxes. Their practical use offers interesting possibilities and often effective props.

1. LR boxes – their content is single-line material typed in a restricted horizontal mode (without the possibility of line breaks). They are created by command

`\makebox[width] [position] {text},`

where the first optional parameter defines the resulting width of the box, the second optional parameter the desired position of the content inside the box (letter l – left, r – right, the default location is centered) and the mandatory parameter contains the own one-line material. What can it be used for? The command is interesting with all the parameters: by specifying the width, we determine how the box should behave towards its surroundings, whatever its content is – we can pretend that the content has a smaller overall width (even zero, for example), or a larger width that will be filled with a space. The way the content is placed then determines where the potential gap will be or on which side the content will be “pushed” out of the box and will not be included in the surrounding typesetting. Prohibiting line breaks in the box material is also very useful in some cases. One common use case is tabular typesetting. Suppose we have a centered column of numeric values that we know must be aligned with the same order below each other. However, if the values are equipped with some other characters (referencing asterisk, plus or minus sign, etc.), they cannot be accepted during alignment,

because they would move the value to an unwanted position. The solution is a zero-width box whose content is the appropriate character:

Simple centering:		Centering with character treatment:	
Yield [k€]	(Source text:)	Yield [k€]	(Source text:)
18,5	18,5	18,5	18,5
-11,9	\$-\$11,9	-11,9	<code>\makebox[0pt][r]{-\$-\$}11,9</code>
16,3*	16,3*	16,3*	<code>16,3\makebox[0pt][l]{*}</code>
...		...	

* Moving average

The second example of use is to create a table with the required column width. Headings are made up of boxes that have a specified width (22 mm for First Name, 32 mm for Surname and 40 mm for Signature):

Attendance List			
No.	First Name	Surname	Signature

If we only need to disable line breaks in a certain part of text and do not want to manipulate the resulting width of the box, we can use the shortened command

`\mbox{text}.`

In addition, when used in a mathematical environment, its property will be reflected – plain text type, using the font set before the math environment.

2. Box framing – a frame can be applied around the box with the \LaTeX command

`\framebox[width][position]{text}`

or a shortened version

`\fbox{text}.`

It is analogous to the commands `\makebox`, or `\mbox`, the parameter meanings are identical. A border is created around the box with a line thickness in the `\fboxrule` registry and the distance of the line from the contents of the box is used from the register `\fboxsep`. Both registers can be set to the desired dimensions without restriction.

3. Rule boxes – these are rectangular areas filled with ink. The basic universal command of the \LaTeX / X_{\LaTeX} system for creating a rule is the command

`\rule[raise]{width}{height}.`

Its optional parameter indicates the position of the bottom edge relative to the baseline (positive values are up, negative values are down), default is zero. Example:
`\rule{20mm}{0.5pt}\rule{10pt}{1em}\rule[0.7em]{20mm}{0.5pt}%
\rule[-0.5em]{2pt}{1.5em}`
yields:



In addition to the mentioned command, variants of \TeX can also be used: commands `\vrule` and `\hrule`. They implicitly create a line with a thickness of 0.4 pt and the font size (`\vrule`), respectively about typesetting width (`\hrule` – in vertical mode only). If we would like to influence their implicit dimensions, we can use the keywords `width`, `height` and `depth`, for example:

`\vrule width 1pt depth 5pt yields |`
`\hrule width .3\linewidth height 1pt depth 1pt yields:`

Line boxes with zero width (so called struts) are widely used. They are invisible, but can be used wherever it is necessary to increase the space derived from the size of the text – in tables, in mathematical expressions, sometimes even inside paragraphs. Examples:

Increase the distance of the text from the bounding lines in the table:

```
\begin{tabular}[t]{|l|c|} \hline
Method: & \begin{tabular}[t]{@{}l@{}}
a) initialization\\
b) values inserting\\
c) sorting
\end{tabular} \\ \hline
Properties: & stability \\
& & sequentiality \\ \hline
\end{tabular}

\def\hstrut{\vrule height 1.2em width 0pt}
\def\dstrut{\vrule depth 0.8em width 0pt}

\begin{tabular}[t]{|l|c|} \hline
Method: & \begin{tabular}[t]{@{}l@{}}
a) \hstrut initialization\\
b) values inserting\\
c) \dstrut sorting
\end{tabular} \\ \hline
Properties: & \hstrut stability \\
& & \dstrut sequentiality \\ \hline
\end{tabular}
```

Method:	a) initializing b) values inserting c) sorting
Properties:	stability sequentiality

Method:	a) initializing b) values inserting c) sorting
Properties:	stability sequentiality

Improving readability of mathematical expression:

```
\def\hstrut{\vrule height 1.1em width 0pt}
\def\mstrut{\vrule height 0.75em depth 0.3em width 0pt}
\def\dstrut{\vrule depth 0.5em width 0pt}
 $\sqrt{2}$   $\frac{2}{\sqrt{a^2}}$   $\sqrt{2} \frac{2}{\sqrt{a^2}}$ 
 $\sqrt[2]{2}$   $\frac{2}{\sqrt[2]{a^2}}$ 
 $\sqrt[2]{2}$   $\frac{2}{\sqrt[2]{a^2}}$ 
 $\sqrt[2]{2}$   $\frac{2}{\sqrt[2]{a^2}}$ 
```

- Paragraph boxes – their content is in restricted vertical mode, line breaks are allowed, but not page breaks. A command available in \LaTeX / X_{\LaTeX} is

`\parbox[ref. point]{width}{material}`

or environment

```
\begin{minipage}[ref. point]{width}.
```

The position of the reference point in the optional parameter is set by the letter t (top) = baseline of the first line of the box content or b (bottom) = bottom edge of the box. By default, it is in the vertical center.

Mandatory parameter specifying width determines the width of the content of the box, by default it is justified. With the command `\raggedright`, `\raggedleft` or `\centering` inside the box, the alignment can be adjusted to the left, right or center.

Environment `minipage` behaves identically, its advantage is a different delimitation of the contents of the box, which can be divided for example into two macros (start part, end part), while `\parbox` must always be specified in its entirety inside the one macro.

The use of paragraph boxes is wide. They significantly expand the possibilities of tables (the table field can be in the form of a paragraph), but also allow different composition of elements, for example, two images with captions next to each other, keeping the material on the same page (no page break), etc.

Example of a table with three equally wide columns with paragraph content:

Table with paragraph fields

```
\def\tabf#1{\parbox[t]{.25\linewidth}{\raggedright #1}}
\begin{tabular}{|l|l|l|} \hline
\tabf{\bfseries Lorem ipsum dolor sit amet consectetur Cum.} &
\tabf{Condimentum ligula ante Phasellus lacinia ut Integer
Vestibulum Mauris lacinia Nullam. Justo orci dolor justo.} &
\tabf{In molestie mauris vel non volutpat faucibus et magnis
id netus sit nunc. Euismod quis Vestibulum.} \\ \hline
\tabf{\bfseries Condimentum ligula ante Phasellus} &
\tabf{Lorem ipsum dolor sit consectetur. Cum lacinia ut Integer
Vestibulum Mauris lacinia Nullam. Justo orci dolor justo.} &
\tabf{In molestie mauris vel non volutpat faucibus et magnis
id netus sit nunc. Euismod quis Vestibulum.} \\ \hline
\end{tabular}
```

After compilation we get:

<p> Lorem ipsum dolor sit amet consectetur Cum.</p>	<p>Condimentum ligula ante Phasellus lacinia ut Integer Vestibulum Mauris lacinia Nullam. Justo orci dolor justo.</p>	<p>In molestie mauris vel non volutpat faucibus et magnis id netus sit nunc. Euismod quis Vestibulum.</p>
<p> Condimentum ligula ante Phasellus</p>	<p>Lorem ipsum dolor sit consectetur. Cum lacinia ut Integer Vestibulum Mauris lacinia Nullam. Justo orci dolor justo.</p>	<p>In molestie mauris vel non volutpat faucibus et magnis id netus sit nunc. Euismod quis Vestibulum.</p>

5. Basic \TeX commands for boxes – they work in all cases, this is the command `\hbox{content}`, or `\hbox to measure{content}` for restricted horizontal mode and command `\vbox`, respectively `\vbox to measure{content}` for restricted vertical mode.

11.2.5 Affecting the base typesetting method

As already mentioned, the basic typesetting method consists of setting consecutive boxes one after together with reference points on the baseline. In addition, spaces can be inserted (already mentioned in several places) and penalties affecting the line break algorithm. This activity can be influenced by the following commands:

- The next box will not be right behind the previous one, but will be shifted horizontally – command

`\kern measure`

If the *measure* is negative, the next box is inserted over the content of the previous box. It is a command that explicitly solves the so-called kerning, but it can be used for any other similar purpose.

Example: Source code `\kern-5.5ex{//////// gets/et/ot}`

- The box will not be placed on the baseline with its reference point, but above/below – in the \LaTeX / \XeLaTeX system the command

`\raisebox{raise}[height][depth]{text}`

A LR box will be created with the given *text*, whose reference point will be positioned according to the *raise* parameter (positive values indicate the direction up, negative values down). The total height and depth of the resulting box can then be affected by the corresponding optional parameters. So, for example, it can be pretended that even if something is increased/decreased, it should not be included in the total height/depth of the line by its natural size, but by some other (bigger, even smaller or zero).

Examples: Consider two definitions of command `\th`:

1. `\def\th{\raisebox{.5em}{\scriptsize th}}`
2. `\def\th{\raisebox{.5em}[0pt]{\scriptsize th}}`

We will use them in the same material and observe what results we get:

... It was presented as a 15`\th` contribution in the 4`\th` section.

Definition 1 (broken line spacing):
Part of the design management of the large track. It was presented as a 15th contribution in the 4th section.

Definition 2 (balanced line spacing):
Part of the design management of the large track. It was presented as a 15th contribution in the 4th section.

- Affecting line breaks – very often we need different kinds of solid spaces (1/4 em, 1/6 em), which we can define as a regular space of the given size, but with a “penalty” for the line break at that point. If we want to completely disable the break, we set this penalty with the command `\penalty` to the highest possible value, i.e. 10,000.

Examples:

– Solid space 1/4 em: `\def\,{\penalty10000\hspace{0.25em}}`, we will use it, for example, in conjunction `Dr.\,Stark` – Dr. Stark

– Solid space 1/6 em: `\def\;{\penalty10000\hspace{0.1667em}}`, we will use it to space the ellipsis `\dots`; and yet it spins! – ... and yet it spins!

11.2.6 Graphics operations

The same operations can be performed with any box (with `graphicx` package), the effect of which is also incorporated into the given command for inserting a graphic file. It’s resizing, mirroring and rotating. The following commands are available:

- `\scalebox{scale-x}[scale-y]{content}` – command scales its single-line *content* by a factor of *scale-x*. If the optional parameter is not specified, this scale is also used in the *y* axis and the scaling is proportional to the original size ratio. Examples:


`\scalebox{3}{\&}` – &
`\scalebox{3}[1]{\&}` – &


- `\reflectbox{content}` – the command performs mirroring in the *x* axis, for example `\reflectbox{\&}` – ⌘. The same effect can be achieved by writing `\scalebox{-1}[1]{\&}`. It also follows that we write the mirroring in the *y* axis (and at the same time the change in size), for example: `\scalebox{2}[-2]{\&}`

– ⌘

- `\resizebox{dimension-x}{dimension-y}{content}` – command to resize to appropriate dimensions in both axes. If we don’t want a disproportionate change, we write only the dimension in one axis and instead of the second dimension just

the character !. Dimensional values can of course be written using length registers and coefficients when we need for example to adjust table to typeset width. Examples:

`\resizebox{2.5em}{!}{\&} -` 





`\resizebox{2em}{3\baselineskip}{\&} -` 

- `\rotatebox[parameters]{angle}{content}` – command rotates the *content* by the specified angle. The angle is implicitly in degrees and the rotation is counterclockwise. Optional *parameters* are written similarly to inserting images in the form *key=value* and can be the following options:

origin=center of rotation – given as one or two letters from the list: lrctbB in the usual meanings (l – left, etc.). The letter B stands for baseline.

x=dimension, y=dimension – determining the center of rotation using relative coordinates from the current reference point.

units=number – allows two settings: the number -360 changes the direction of rotation, the number 6.283185 changes degrees to radians (= 2π). Examples:

`\rotatebox{90}{...y...} -` 
`\rotatebox[origin=t]{90}{...y...} -` 
`\rotatebox[x=0.9\width, y=0.9\height]{90}{...y...} -` 
`\rotatebox[units=-360]{90}{...y...} -` 

11.3 Inserting images in a document

The floating environment `figure` is used to include images (or any other named object) in the document. It is (except for the name) identical in behavior to the already described environment `table`. It forms its own series of floating objects that can be alternate with the tables, but the order of the objects themselves `figure` remains the same as the order in the source text. The command `\caption` may also appear here with the label, the environment has its default counter `figure` and can be saved into labels and used in cross-references. Similar to the `table` environment you can create a list of figures with the command `\listoffigures`.

11.4 Creating pictures in the system – environment `picture`

In some simpler cases, we don't need to rely on graphic operations handled by the output driver or some external image editor, but we can create an image using text tools working on the same principle as normal paragraph text – inserting characters of a certain font at the typeface position.

This purpose is served by the special system environment \LaTeX / X_{\LaTeX} `picture`. The basic idea is to compose selected image components from characters of special fonts, namely at positions given by coordinates on the specified desktop. Since there cannot be an infinite number of elemental elements in fonts, there are some restrictions on shapes, slopes and sizes in some cases, but still the common needs of simple graphics are covered. In addition, the very principle of arbitrary placement of the “typesetting point” can be used, thus achieving effects unavailable with other tools.

All dimensions written in this environment and in its individual commands are entered as relative values multiplied by the value in the length register `\unitlength`. Its default value is 1 pt.

Environment exception `picture` it is also emphasized by a slightly different syntax. The entries of coordinates and some dimensions are surrounded by round brackets, but the principle of optional parameters in square brackets and other mandatory parameters in brackets remains.

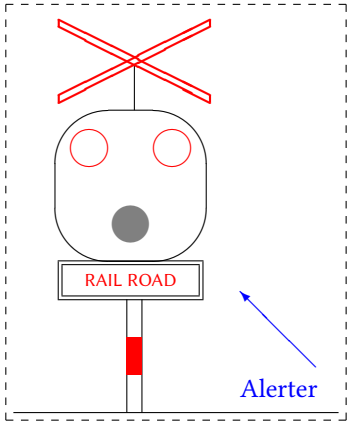
We will briefly describe the basic features and options using a short example:

Example of `picture` environment

```

\unitlength=1mm % for better idea about dimensions
\def\konec{\color{red}\line(0,1){1}}
\begin{picture}(43, 50) %canvas -- box
\put(-1,-1){\dashbox(45, 55)} %extends -- never mind
\put(10, 35){\color{red}\circle{5}} %dimension limit
\put(22, 35){\color{red}\circle{5}}
\put(15.5, 25){\color{gray}\circle*{5}}
\put(15.5, 30){\oval(20,20)} %predefined arcs
\put(15, 0){\line(0,1){15}}
\put(17, 0){\line(0,1){15}}
\put(15, 5){\color{red}\rule{2mm}{5mm}} %simple "text"
\put(6, 15){\framebox(19, 5){%
  \framebox(18,4){\color{red}%
    \scriptsize\sffamily RAIL ROAD}}}
\put(16, 40){\line(0,1){6}}
\thicklines
\put(6, 41){\color{red}\line(2,1){20}} %slopes
\put(6, 42){\color{red}\line(2,1){20}}
\put(6, 51){\color{red}\line(2,-1){20}}
\put(6, 52){\color{red}\line(2,-1){20}}
\put(6, 41){\konec} %the use of macros
\put(6, 51){\konec}
\put(26, 41){\konec}
\put(26, 51){\konec}
\thinlines
\put(0,0){\line(1,0){43}}
\put(30, 2){\color{blue}Alerter} %simple text
\put(40, 6){\color{blue}\vector(-1, 1){10}}
\end{picture}

```



As you can see from the example, the environment contains commands for rendering certain image objects and the possibility of placing their reference point in any place. The command

$\put(x, y)\{object\}$

places *object* anywhere on the desktop, but also outside it – the affiliation of the coordinates x and y to the workspace space is not checked in any way.

The second command to place the object is

`\multiput(x, y)(\Delta x, \Delta y){number}{object}`.

The command performs the placement of the corresponding *object* starting from the first coordinates as many times as the parameter *number* indicates, while changing the position of the placement by the difference Δx and Δy .

The simplest object is text, its reference point is on the left edge on the baseline. The command `\rule` can also serve as text to draw a line or rectangle. Furthermore, the object for drawing the circle `\circle` is used in the sample or circle `\circle*`. As already mentioned, arcs and slanted lines are composed of corresponding elements in the form of letters of a special font. Therefore, there is a limitation on the size of both the circle and the circle* (diameter approx. 10 mm or 6 mm).

The `\line` and arrows `\vector`. are somewhat more complicated to enter. The lengths of the segments are entered in round brackets, so for example (2, 1) means segments with a projection of two units to the x axis and one unit to the y axis, so it is a line with a slope of about 26° (exactly it is $\arctan \frac{1}{2}$). Combinations can contain the values ± 1 to ± 6 . If one value is zero, then the other value is always 1 and yields a horizontal or vertical line. Negative value represents the opposite quadrant.

Circles, ovals, and slanted lines can be drawn in two thicknesses – 0.4 pt (default) or 0.8 pt when toggled with `\thicklines`. You can switch from thick back to thin with `\thinlines`.

Any part of the drawing can be rendered in any color, similar to normal text (command `\color`).

An interesting object is `\framebox`, whose parameter may contain some other object – in the example it is text. It is implicitly placed in the center of the framed box. However, the command also has an optional parameter, where the letters lrtbc can specify other of the nine possible locations, eg the combination lt defines the upper left corner (left, top), etc. Not specifying a letter means an implicit center. The dashed bounding box `\dashbox` works similarly and there is also a box without framing `\makebox`, whose primary purpose is the possibility of placing its contents in one of the nine selected points.

A very important application option of the `picture` environment is its use to insert any object in any place in the page area. If we write for example:

```
\begin{picture}(0,0)\put(100,-30){\line(1,0){50}}\end{picture},
```

a box of zero size is created (it doesn't affect where it's written in the typesetting or where its internal graphics objects point at all), but a 50-unit horizontal line appears at a location 100 units horizontally and 30 units `\unitlength` down. For example, clipping marks can be rendered by placing an appropriate sequence of lines in the null environment of `picture` placed in a running heads.

In a similar way, in the example, the image was placed on the right side of the page next to the text with the `picture` environment commands.

Another situation where zero environment dimensions are useful are image labels for image inserted from a external file. We can create an image without labels in some program and add the labels after inserting them into the text in order to achieve a uniform font with the rest of the document. In addition, the image can be used in different documents and always achieve a completely consistent look.

It goes without saying that parts of an image, parts of commands or even the entire environment can be used as the body of the definition of a new command (in the

example the command `\konec = end`), so repeating parts can be very advantageously automated, dimensionally and shape-parametrized, and thereby obtain considerable advantages for any modifications and changes that balance out the somewhat cumbersome and restrictive notation style.

Document as a whole

In the previous chapters, we dealt with the individual elements of the document in turn, starting with the font, special characters, through paragraphs to images and other graphic elements, now we will look at the document as a whole, its page arrangement and some options for transferring information between individual parts.

12.1 Pages arrangement

The order of pages in a document is not random, but like other elements, a certain arrangement has been established over time that the reader expects and that allows him to efficiently find the information he is looking for in the document.

12.1.1 One-sided or two-sided?

For the document, we have to decide whether the result will be a print on paper or we will keep only its electronic form. As mentioned earlier, in the electronic form it is necessary to choose slightly different fonts and other graphic elements (for example, colors). But we also have to choose the page layout – we usually choose a **one-sided** type of document where all pages have the same set margins (even and odd), also the margin settings may be somewhat different from a document printed on paper, because the common format of the screen or projector is landscape, not portrait.

For printed documents, we choose the **two-sided** type – the odd-numbered pages are always on the right, the even pages on the left, and the double-page has symmetrical margins – the size of the left margin on the left page is equal to the size of the right margin on the right page, likewise the inner edges at the spine are identical.

12.1.2 Document arrangement

In documents of different scope and purpose, there are different pages of certain types and in some cases their order also changes slightly, but in general it can be said that the basic principles remain the same. The document that typically has the most page types is a book. In the following list, we will mention commonly used elements in books, but also elements typical for, for example, final or qualification theses. Broadly, document elements can be divided into three parts: everything before the main body is sometimes called the front part, everything after the main body is called the back part. These sections can sometimes be distinguished by different pagination as well as page numbering. For example, it is customary for some publishers to number the front part of the book with Roman numerals and the main part with Arabic numerals, with the numbering of the main part starting again from one.

Cover – it is a visually attractive element, especially for a book, so that the book can be seen from a distance on the bookstore shelf. Quite often, the graphic elements used are different from those used in the document, although a well-designed piece should be visually correspond to the content. (Technical design of the cover – see overview of bindings.)

Inset – a sheet of somewhat stiffer paper, which is used to glue and hang a book block in a hardcover. Most often it is completely empty, but sometimes it is used for additional graphics (children’s books) or is printed with a graphic pattern or other motif.

Foretitle – the name of the document in a basic typeface placed usually in the optical center of the page; introduces the reader to the entire document (used especially for books). The subtitle page is numbered as 1, although the page number is not usually given.

Frontispice – left facing page to main title; it is often empty, but sometimes forms a graphic unit with the main title and contains, for example, a full-page image (photo) or other additional information.

Title – narrative page giving the most important information about the document: title, subtitle, author, year or date of publication, publisher, etc. A properly designed title page uses graphic elements corresponding to the design of the document, both in its layout and font parameters.

Publisher’s record – page with information about the creation of the document; if the document has been assigned a standard number (ISBN), it must be listed here. In older publications, the publisher’s record is very brief and contains the names of the authors, the date of publication and any indication of copyright, in the case of professional publications, the names of reviewers, advisors, etc. In newer publications, the foreign custom of stating the publisher’s information, the names of editors and other persons involved in the creation and production of the work is often adopted here (data from colophon).

Dedication, Acknowledgments, Motto – single pages with usually simple design (basic font size in optical center).

Abstract – especially for professional and final theses, a several-line specification of the content of the work presented in various research library systems and used for quick orientation of the reader. It is available in several languages and is supplemented with key words (passwords) facilitating the definition of the discussed issue.

Contents – a list of important parts of the document (chapters, sections) used to orient the reader in the document. It is usually listed only in professional publications, in fiction (if at all) it is often placed at the back of the book. In the case of more extensive publications, the content can be designed at multiple levels – a coarser (overview) level with a list of only larger units and a subsequent fine (detailed) level with additional information. In this way, the reader can more precisely search for the place he is interested in in the document.

Lists of figures and tables – this is additional information to the content; again, it is important how this information is beneficial for the reader and for his orientation in the document. If listed, they should always speed up the search for a site containing the specific thing the reader is likely to follow.

List of terms, abbreviations – for professional documents, it is suitable if the author wants to precisely define the meanings of terms and abbreviations used in professional practice in multiple meanings, or wants to introduce his own terms and abbreviations to clarify and shorten the interpretation. In this section, terms and abbreviations should not have fixed meanings.

Preface – text containing information about the motive, method of creation and other contexts of the creation of the publication. Other helpers are often mentioned here – people who participated, for example, in data collection and cleaning, laboratory work, professional consultations, etc.

Introduction – Text related to the focus of the publication and often already included in the main part of the publication.

Main body – chapters, sections, subsections – the entire content of the publication.

Afterword – text with the same meaning as the preface, but placed at the back of the publication (then the publication does not have a preface).

References – a list of all sources from which the authors of the publication drew any ideas, inspiration, data and other elements. It is of fundamental importance in professional works and is largely standardized (ISO 690 standard).

Summary – a section with a similar meaning to the abstract, but appearing at the back of the book. The summary is sometimes larger than the abstract and focuses on the results of the work (especially for research, development and final theses).

Index – represents a list of important terms of the entire publication. It is suitable for more extensive and overview documents, textbooks and manuals. A high-quality index can significantly help the reader to find his way around the publication and allow him to quickly find essential information. The proposal of passwords for the register and their arrangement is an important author's work, but at the same time very demanding on accuracy and completeness. There may be more than one index in one publication (subject index, name index, Latin term index, etc.). Indexes created by an editor (typewriter) or completely automatically are usually of unsatisfactory quality and contribute little to the orientation of the reader.

Appendices – parts containing material that does not fit into the main text, but expands the information richness of the publication. Typically, original data from which the publication draws and analyzes it, or complete source codes of developed programs, various lists and additional resources can be inserted into the appendices. The typesetting of the attachments can be done with the same parameters as the main text, only the numbering of the headings changes, but it is also possible to see the attachments in a different design, because it is convenient to preserve the original formatting of the attached resources.

Colophon – summary technical information about the publication, usually located on the last odd page of the publication. Data: complete title with all sub-titles, all authors, editors, illustrators, translators, etc., information on the technical design (in the case of printing about type, printer, circulation, etc.), inclusion of the publication in the edition series, identification numbers, standard number, recommended price. In publications adopting foreign customs, these data are given in the publisher’s record in the front part of the document, while the imprint in the back part is completely missing. Typesetting uses a basic font and its basic or slightly reduced grade, often arranged in the form of a table and placed at the bottom of the page.

12.2 Moving of information

The above overview of parts of the documents mentions the need to obtain and edit information from other parts of the documents. A typical case is the table of contents – a list of chapters, sections, subsections, or other material, indicating the location (page) in the document. Similarly, a list of tables and figures, possibly lists of terms, keywords, etc. Another similar element is links between specified places in the document – links to figures, tables, chapters, sections, footnotes, list items, etc.

All the mentioned problems have one common denominator – the solution of information moving. In previous chapters, we have already stated that this moving is technically realized by storing information in auxiliary files during the compilation of the document, these auxiliary files are then read in the following compilation and the stored data is interpreted and put in the appropriate place.

We have already familiarized ourselves with the auxiliary files `*.toc` (content items), `*.lot` (list of tables), `*.lof` (list of figures) and `*.aux` (cross references). These files are created and used by predefined commands, and we as users no longer have to worry about their handling.

12.2.1 Fragile and robust commands

If we place some text material containing macros in the auxiliary file, it may happen that these macros will be expanded and the file will no longer contain the original macro, but its expanded form. When this file is subsequently read during the next compilation, the expanded form of the macro may no longer be processable. Macros that expand are called **fragile commands** because they “break”. Commands that do not perform this expansion are called **robust**. When defining a custom command (macro), we can specify that the command should be robust if instead of `\newcommand` we use the same syntax `\DeclareRobustCommand`. If we want to prevent the premature expansion of a fragile macro transferred to the auxiliary file, we use the command `\protect` before it or `\TeXform \noexpand`.

12.2.2 Index

The creation of the index is somewhat different from the previous cases of information moving. It is similar to the content, for example, of transferring the index item and the page number of its occurrence to another place in the document, but in addition, the items

need to be organized and sorted alphabetically. This is an operation that must be provided by an external program. Thus, the process of creating the index has the following steps:

1. Creating the auxiliary file `*.idx` with items, their attributes and page numbers during the first compilation. The auxiliary file contains the sequence of macros `\indexentry{item}{page}` in the order in which the commands `\index{item}` appeared in the source text. The auxiliary file is created and filled in if we include the command `\makeindex.` in the preamble of the document.

The command to create a password `\index` has other options in addition to the mentioned basic meaning:

- Determining the sub-item: `\index{item!subitem}`, possibly `\index{item!subitem!subitem}`. It is important that the items at a higher level match exactly (pay attention to inflection, etc.).
- Help for item sorting: `\index{sorting@item}` – example: `\item{betacarotene@β-carotene}`, where we want the item β -carotene to be listed as the text “betacarotene”.
- Distinguishing the importance of item placement: `\index{item|cmd}` – for example `\index{item|textbf}` will generate an index entry where the page number will be inserted as a parameter of the command `\textbf`. We can thus distinguish the place that is more important for the reader (for example, this term is defined here) from the place where the term is only used or is in some a freer relationship.
- Page range: `\index{item|{}` for start and `\index{item|)}` for the end. A page range from the start page to the end page will be generated in the index. This is useful if the given item appears here many times and it would not be beneficial to list several consecutive pages in one index entry.

2. The obtained auxiliary file is converted to the resulting list of items with an external program, the typical extension of the resulting file is `*.ind`. In this file, the sequence of commands is `\item` with items text and links to pages; if the index has subitems, there are commands `\subitem`, or `\subsubitem` for two nested levels. Another command generated to the index is `\indexspace` defining the vertical space separating the individual letters of the register.

An external program for processing registry items is, for example, `Makeindex`, `CSIndex` is available for the Czech and Slovak locales. However, it only uses one-byte encoding of national characters. If the items are in UTF-8 encoding, it is necessary to use, for example, the program `xindy`. All programs are usually represented in common distributions and have their own documentation.

3. Custom typesetting of the index is created by the environment `\begin{theindex} ... \end{theindex}`, the calculated index in the file `*.ind` is inserted into this environment by the command `\input{...ind}`. By default, items are arranged in a two-column format. To modify the shape of the index, both the `theindex` environment and the `\item` commands etc. can be redefined.

12.2.3 References – bibliographic citations

The valid ISO 690 standard determines the main principles for creating a list of bibliographic citations and references in the text. However, it does not define the exact format used in individual items, nor the exact way of referencing in the text. A number of interpretations of the standard are based only on the examples and subjects that are given in the standard, but are not binding and in some cases do not even correspond to spelling or typographic principles.

The basic purpose of bibliographic citations is to clearly identify borrowed ideas and materials and to distinguish them from the author's own texts. This purpose can then be fulfilled in various ways and with corresponding tools.

A list of bibliographic citations can be thought of quite simply like any other numbered or unnumbered list. It lists individual data about the sources, according to purpose, level of detail, etc. The forms of these data, their arrangement and appearance depend to a large extent only on the decision of the author. What is important, however, is that the list is constantly consistent with the references in the text.

In its informative Appendix A, the standard lists three options for citing sources in the bibliographic citation list:

- Form name–date (Harvard system)

In the text, the reference is written in round brackets with possible page designation, e.g. (Smith and Watt, 2019, p. 243). The list of bibliographic citations is arranged alphabetically by surname so that the reader can find each reference accordingly. No numbering is necessary for the list; if mentioned, it has no meaning in terms of references and citations. In addition, this form is very flexible – the necessary information can be given in the text so that it is a natural part of the sentence, for example “In research by Smith and Watt (2019) it is stated on p. 243 that ...”

Because links can take many forms, there is no simple system that supports all possible forms. At the same time, it is relatively easy to maintain consistency between references and citations: adding a reference places the bibliographic item in the corresponding place in the list, and removing the link removes the list item just as easily. Although these are manual edits, they are always one-time and are not affected by other links and items.

- Form of numerical references

References in the text are in the form of a serial number written in round brackets, square brackets or in the form of an exponent, including additional information (page numbers, etc.). Example: “Research (34, p. 243) states that ...” or “Research^{34, p. 243} states that ...”

The list of bibliographic citations is sorted by the order of occurrence of citations (not alphabetically!), the first reference is always number 1.

In this case, it is necessary that the correspondence between the reference number and the order in the list is ensured automatically. It is the only form that is almost fully supported by the corresponding commands. The command system `\cite{label}` is used for this for references in the text and the list of bibliographic citations in the `thebibliography` environment. In the predefined form, square

brackets are used for references (the standard allows them, but they can be easily redefined to be round or exponent-shaped by modifying the “internal” command `\@cite`). The necessary link between the link and the corresponding item in the list is then formed by the selected label.

The list of bibliographic citations is handled by the aforementioned environment, in which the items defined by the command `\bibitem{label}` data are located. The environment `thebibliography` also has a parameter indicating the required size of sequence numbers so that the list is precisely formatted starting from the first item.

In the following example, only selected items are listed for clarity (missing data are shown by ellipsis). For bibliographic data there are still other simplifying commands – `\litauthor` and `\litname` defining the shape of important items in citations (the author is typeset in small caps, the name in italics). Ensuring the uniformity of the appearance of items is one of the principles defined by the bibliographic standard.

— Example of bibliographic citation —

```
\begin{thebibliography}{00} % less than 100 items in the list are assumed
\addcontentsline{toc}{chapter}{References}
\markboth{References}{References}
...
\bibitem{latex} \litauthor{Lamport, L.} \litname{\LaTeX} -- a Document
  Preparation System: User's Guide and Reference Manual}.
  Reading: Addison-Wesley Publishing Company, 1994. % item in order #8
...
\bibitem{companion} \litauthor{Goosens, M., Mittelbach, F., Samarin, A.}
  \litname{The \LaTeX} Companion}. Reading: Addison-Wesley, 1994. % #35
...
\bibitem{matsazba} \litauthor{Nohel, J.} \litname{Sazba matematická
  a chemická}. Praha: SNTL, 1976. % #51
...
\end{thebibliography}
```

The initial statement opening the bibliographic citation environment produces an unnumbered top-level heading in the given document class (chapter in the book and report classes, section in the article class). The heading text is stored in the macro `\refname` – by redefining it, the text can be modified in any way. Because the heading is unnumbered, it is not automatically included in the table of contents or inserted into regular headings. This can be done by corresponding commands placed just after the beginning of the environment as is shown in our example. Individual items are numbered using the counter `enumiv`, the value of which can be referred to by labels, or its value can be changed as needed. The inserted bibliography will then have the following form in this document:

Bibliography

- [8] ...
LAMPART, L. *LaTeX – a Document Preparation System: User’s Guide and Reference Manual*. Reading: Addison-Wesley Publishing Company, 1994.
- [35] ...
GOOSENS, M., MITTELBACH, F., SAMARIN, A. *The LaTeX Companion*. Reading: Addison-Wesley, 1994.
- [51] ...
NOHEL, J. *Sazba matematická a chemická*. Praha: SNTL, 1976.
...

References are then given in different forms in the text, for example: `\cite{latex}` – yields output as [8], `\cite[s.\,221]{companion}` – is output as [35, s. 221], `\cite{latex, matsazba}` – is output as [8, 51], etc.

The BibTeX system can be used to further automate the entire process. The principle consists in the gradual (and often long-term) building of a database of bibliographic resources (a text file, e.g. `references.bib` of a special format). If there are references to these sources in the text of the document, the external program BibTeX will extract them from the auxiliary file `*.aux`, according to them, it will select the resulting file `*.bbl` the resulting items from the database, it usually also sorts them alphabetically and formats them into items `\bibitem`. This file also contains the environment `thebibliography`, just insert it into the document with the command `\include`. More information to the extensive and multifaceted BibTeX system can be found in the standard documentation in the distribution.

- Form of running footnotes

This form assumes that bibliographic citations are not listed in a single list somewhere at the back of the document, but are continuously placed in footnotes. Each reference in the text is therefore technically solved by the corresponding command `\footnote`, in the text of which there is (at the first occurrence) a complete set of data about the given source, including possible page location. If we refer to such a resource somewhere else, another note number is generated again, but in the text of the footnote you can refer to the footnote number where the previous occurrence is located. Quite ordinary pairs of `\label{label}` and `\ref{label}` can be used for this.

The advantage of this form is undoubtedly the quick availability of data on sources for readers. On the other hand, however, the information is scattered throughout the document, with backlinks to other notes, the reader has to scroll through to find where the note occurred, etc. The system is particularly useful when the references are relatively few and the summary list of citations does not bring any added value to the reader.

In pedagogical theory, it is stated that the optimal success rate of the test is around 50%.¹ On the other hand, Maňák² states that the results of intentional action should be expressed by a fundamentally greater success rate. In addition, it can be noted that Chráska³ here mostly considers entrance tests in which the intentional action is not yet manifested to the necessary extent.

After compilation we get the following result:

In pedagogical theory, it is stated that the optimal success rate of the test is around 50%.¹ On the other hand, Maňák² states that the results of intentional action should be expressed by a fundamentally greater success rate. In addition, it can be noted that Chráska³ here mostly considers entrance tests in which the intentional action is not yet manifested to the necessary extent.

12.3 Production of a physical document

The production of more narrative documents (brochures, books, magazines) is largely a complex production process that the average computer user cannot access. One of the most complex documents commonly produced with office equipment can be a notebook (jotter), the pages of which are held together with a paper clip, or a ring binder or stapler clip. Printing must also correspond to this procedure – in office practice, A4 format papers are most often used, from which an A4 brochure can be created with a clip or ring binding, or a notebook binding with A5 format (two pages on an A4 sheet).

Low print run (up to dozens of pieces) is usually done on laser (black and white and color) printers. Smaller numbers of pieces can also be produced on inkjet printers, but the print may be damaged by moisture or bleed if lower quality paper is used.

In some cases, we need to prepare a document in such a way that we want to change the order of pages, their placement on a sheet of paper, margins for binding and possible trimming, etc. The following subsection deals with this problem.

12.3.1 Imposition pages for printing

In the professional production of a publication (brochure, book), printing is usually done on a much larger paper format than the final publication format – it is economical and the subsequent bookbinding process is considerably easier than if the printing was done directly on sheets of the final size. It is typically printed double-sided on sheets, from which 16-page folders are then folded with three folds. Individual pages must then be imposed on this sheet so that after bending, folding and cutting, a 16-page “brochure” is

¹CHRÁSKA, M. *Didactic tests*. Prague: SPN, 1987, p. 46.

²MAŇÁK, J. *Outline of Didactics*. Brno: Paido, 1998, p. 118.

³See note 1, p. 81.

automatically created – see Fig. 12.1. These folders are then placed together and glued in the spine, creating the entire publication.

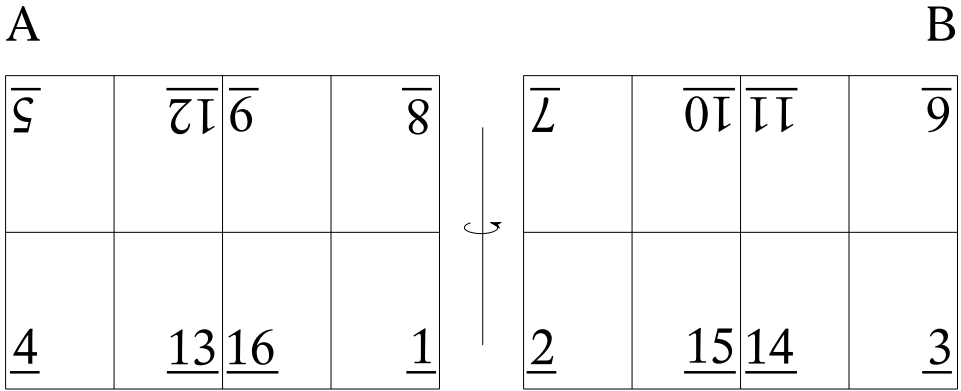


Figure 12.1: Imposition on the sheet of 16 pages (A – face of the sheet, B – back of the sheet)

In the case of simpler printed materials with a booklet binding, the imposition gang stitched, the individual sheets are printed on both sides with 4 pages, the sheets are placed on top of each other and folded with one fold. In the place of fold, the publication is stitched together. The layout of the pages for an example of a 16-page publication is shown in Fig. 12.2.

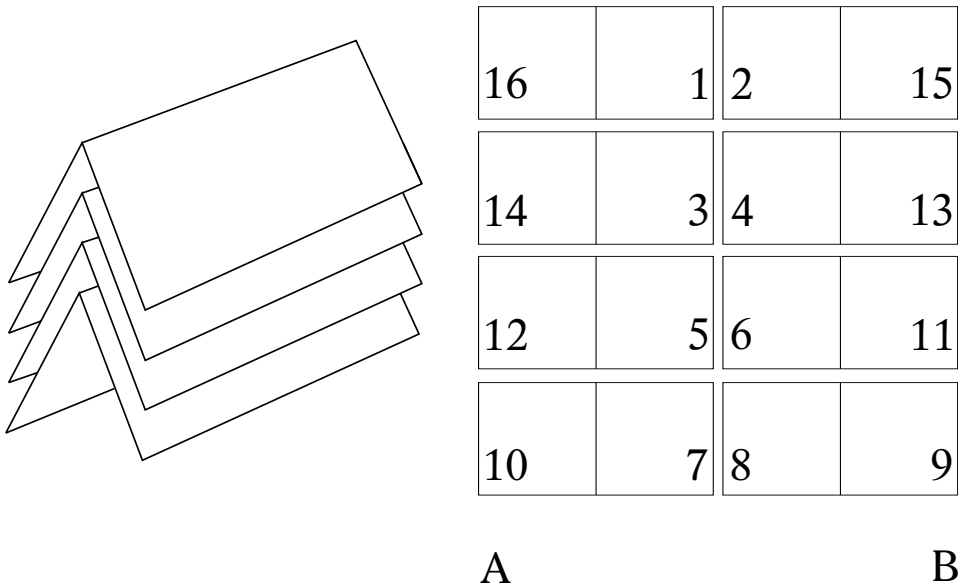


Figure 12.2: Imposition for gang stitching for 16-page publication (A – face of the sheet, B – back of the sheet)

Printer houses use complex and expensive programs to impose pages and many other essentials. On the other hand, a simple A5 brochure with a gang stitching can

be quite easily made into A4 office paper. To imposition pages, we can use commonly available programs:

- There is a `pdfpages` package with a number of parameters for processing the finished typesetting in PDF. We create a source file in which we connect this package and insert it with the command `\includepdf [options]{file}` a link to a PDF publication created in the usual way. The options allow you to control the way individual pages are inserted and their distribution on sheets. As one of the most common examples, we can mention the imposition of pages when printing two A5 pages on A4 format sheets (fig. 12.2). Let's assume that the finished typesetting is ready in the file `info.pdf`. We will prepare the following document:

```
\documentclass{article}
\usepackage{pdfpages}
\begin{document}
\includepdf [pages=-, angle=90, booklet, openright]{info.pdf}
\end{document}
```

By compiling to PDF output, the pages will be imposed into a brochure.

- Imposition of pages during printing can also be done in a PDF format file browser, for example Acrobat Reader. It allows you to rearrange the pages in order for a booklet, for example, but there are no options to influence the scale, margins and offset on the paper so that the pages are in the optimal place after folding and stapling into a V1 binding.

12.3.2 Overview of bookbindings

The binding of the document is important during creation so that we can correctly design the placement of the typeset material on a sheet of paper. Each binding takes some space from the surface of the paper – at the spine, where the papers are joined (glued, stapled) and at the edges, where trimming is done. Approximately 5 mm is calculated for the cut, the loss on the back side is different. Let's take a look at common bindings and their effect on available paper area.

We divide book bindings into three categories: flexible (marked V1–V4), semirigid (V5, V6) and hard-case (V7–V9). Next, we will focus more on flexible bindings, because they are most often considered when creating office printed materials. Semirigid V5 binding (cardboard cover and sewn book block) is practically no longer produced, V6 is folding picture book. Hard-case bindings are exclusively the domain of professional production – they are durable book bindings with sewn folders mounted with board paper into a hard envelope covered with paper, book cloth or leather.

Flexible bindings are (see Fig. 12.3):

- V1 – back-wire stitching. The folders gang-stitched and joined in the middle with staples. Suitable for a maximum of 60 sheets (120 pages). In the spine, you can count on minimal space loss (up to 5 mm).
- V2 – perfect binding (also paperback). The book block is glued in the spine and pasted into the cover with. The cover is also partially glued from the side, this is a space that cannot be used on the paper – approx. 10 mm, sometimes more.

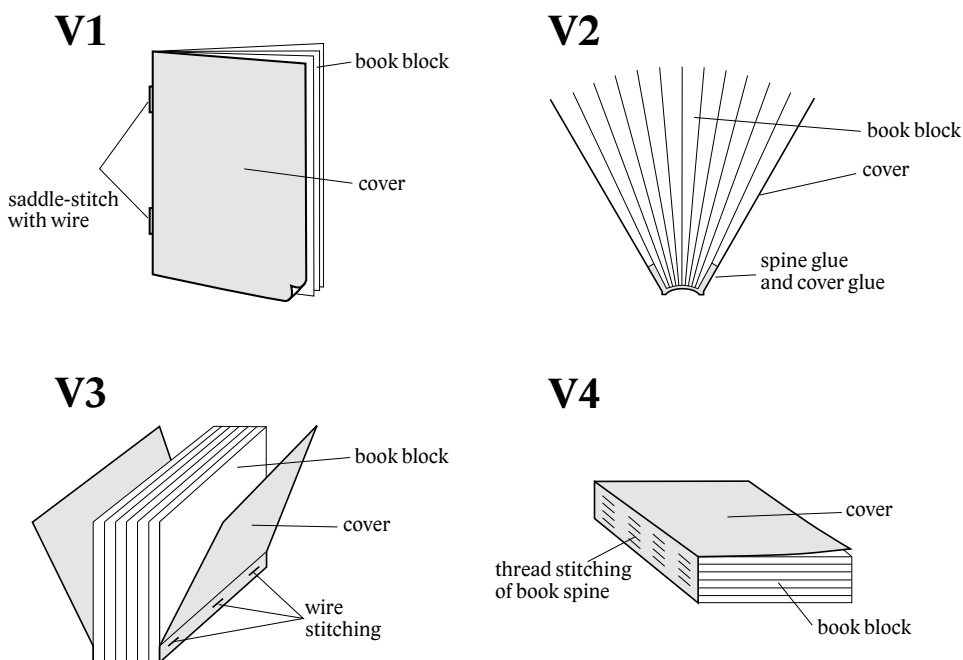


Figure 12.3: Overview and schemas of flexible bindings

- V3 – stitched binding. The folders of the book block are connected to the cover with staples from the side. Considerable space is lost in the spine, approx. 20 mm. With a larger number of sheets, the publication is difficult to open, and the space near the spine is difficult to see.
- V4 – stitched binding. The folders are stitched with threads, the entire book block is glued on the spine and stuck in a cover. The binding is of relatively high quality, the publication is it opens relatively well and the space in the back is easily accessible. Can count with a loss of approx. 5 mm.

Each of the listed bindings can also have a different design of the cover. It is usually a heavier weight paper, sometimes it can have a different color. It is supposed to be printed with the title information (on the first cover page) and sometimes the colophon or just the sales barcode on the last page. For cost-saving reasons, the inner pages of the envelope are sometimes used for additional information (publisher's record on the second page, or colophon on the third page of the cover).

According to the intended binding, we adjust the edges and other page elements, or adjust the parameters when imposing the pages on the sheet. The total number of pages of the publication must be completed on whole sheets. For example, if it is a V1 binding of A5 format and we are preparing A4 sheets printed on both sides, the number of pages of the publication must be divisible by four. We will either edit the document so that it takes up more pages as needed, or we will insert flyleaves in the appropriate places. At the same time, it is necessary to ensure that important information is always on the right (odd) pages.

In professional processing in bookbindings, the appropriate number of flyleaves is usually added automatically at the end of the publication when the pages are imposed.

References for further study

- GOOSENS, M., MITTELBAACH, F., SAMARIN, A. *The L^AT_EX Companion*. Reading: Addison-Wesley, 1994.
- ISO ISO 80000. Quantities and units. Part 1: General. ISO, 2009.
- ISO ISO 80000. Quantities and units. Part 2: Mathematical signs and symbols to be used in the natural sciences and technology. ISO, 2009.
- KNUTH, D. E. *The T_EXbook*.
- LAMPORT, L. *L^AT_EX – a Document Preparation System: User’s Guide and Reference Manual*. Reading: Addison-Wesley Publ. Comp., 1994.
- NOHEL, J. *Sazba matematická a chemická*. Praha: SNTL, 1976.
- OETIKER, T., SERWIN, M., PARTL, H., HYNA, I., SCHLEG, E. *The Not So Short Introduction to L^AT_EX Or L^AT_EX in 280 minutes*. October 11, 2022, <https://tobi.oetiker.ch/lshort/lshort.pdf>
- OLŠÁK, P. *T_EXbook naruby*. Brno: Konvoj, 2002.
- OLŠÁK, P. *T_EX in a Nutshell*,
<https://www.cstug.cz/bulletin/pdf/2021-1-4.pdf>,
<https://petr.olsak.net/ftp/olsak/optex/tex-nutshell.pdf>,
Version of the text: 0.8+ (2022-06-10)
- OLŠÁK, P. *Typografický systém T_EX*. Praha: Grada, 2001.
- RYBIČKA, J. *L^AT_EX pro začátečníky*. Brno: Konvoj, 2003.
- SATRAPA, P. *L^AT_EX pro pragmatiky*. Verze: 1.1, 2011, <http://www.nti.tul.cz/~satrapa/docs/latex/latex-pro-pragmatiky.pdf>

Appendices

Overview of elements used in mathematical environments

Spaces

<code>xx</code>	without space	<code>x x</code>	<code>\,</code>	thin space
<code>xx \!</code>	negative narrow space	<code>x x \:</code>	<code>\:</code>	middle space
<code>x x \;</code>	wide space	<code>x x \quad</code>	<code>\quad</code>	quadrat (em)
<code>x x _</code>	interword space	<code>x x \quad\quad</code>	<code>\quad\quad</code>	two quadrats

Greek alphabet

α	<code>\alpha</code>	θ	<code>\theta</code>	o	<code>o</code>	τ	<code>\tau</code>
β	<code>\beta</code>	ϑ	<code>\vartheta</code>	π	<code>\pi</code>	υ	<code>\upsilon</code>
γ	<code>\gamma</code>	ι	<code>\iota</code>	ϖ	<code>\varpi</code>	ϕ	<code>\phi</code>
δ	<code>\delta</code>	κ	<code>\kappa</code>	ρ	<code>\rho</code>	φ	<code>\varphi</code>
ϵ	<code>\epsilon</code>	λ	<code>\lambda</code>	ϱ	<code>\varrho</code>	χ	<code>\chi</code>
ε	<code>\varepsilon</code>	μ	<code>\mu</code>	σ	<code>\sigma</code>	ψ	<code>\psi</code>
ζ	<code>\zeta</code>	ν	<code>\nu</code>	ς	<code>\varsigma</code>	ω	<code>\omega</code>
η	<code>\eta</code>	ξ	<code>\xi</code>				

Γ	<code>\Gamma</code>	Λ	<code>\Lambda</code>	Σ	<code>\Sigma</code>	Ψ	<code>\Psi</code>
Δ	<code>\Delta</code>	Ξ	<code>\Xi</code>	Υ	<code>\Upsilon</code>	Ω	<code>\Omega</code>
Θ	<code>\Theta</code>	Π	<code>\Pi</code>	Φ	<code>\Phi</code>		

Arrow symbols

Symbols marked in the following tables with the ⁺ tag are available through the `latexsym` package.

\leftarrow	<code>\leftarrow</code>	\longleftarrow	<code>\longleftarrow</code>	\uparrow	<code>\uparrow</code>
\Leftarrow	<code>\Leftarrow</code>	\Longleftarrow	<code>\Longleftarrow</code>	\Uparrow	<code>\Uparrow</code>
\rightarrow	<code>\rightarrow</code>	\longrightarrow	<code>\longrightarrow</code>	\downarrow	<code>\downarrow</code>
\Rightarrow	<code>\Rightarrow</code>	\Longrightarrow	<code>\Longrightarrow</code>	\Downarrow	<code>\Downarrow</code>
\leftrightarrow	<code>\leftrightarrow</code>	\longleftrightarrow	<code>\longleftrightarrow</code>	\updownarrow	<code>\updownarrow</code>
\Leftrightarrow	<code>\Leftrightarrow</code>	\Longleftrightarrow	<code>\Longleftrightarrow</code>	\Updownarrow	<code>\Updownarrow</code>
\mapsto	<code>\mapsto</code>	\longmapsto	<code>\longmapsto</code>	\nearrow	<code>\nearrow</code>
\hookrightarrow	<code>\hookrightarrow</code>	\hookrightarrow	<code>\hookrightarrow</code>	\searrow	<code>\searrow</code>
\leftharpoonup	<code>\leftharpoonup</code>	\rightharpoonup	<code>\rightharpoonup</code>	\swarrow	<code>\swarrow</code>
\leftharpoondown	<code>\leftharpoondown</code>	\rightharpoondown	<code>\rightharpoondown</code>	\nwarrow	<code>\nwarrow</code>
\rightrightarrows	<code>\rightrightarrows</code>	\leadsto	<code>\leadsto</code> ⁺		

Relational symbols

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

Binary operators

\pm	<code>\pm</code>	\cap	<code>\cap</code>	\diamond	<code>\diamond</code>	\oplus	<code>\oplus</code>
\mp	<code>\mp</code>	\cup	<code>\cup</code>	\triangleup	<code>\triangleup</code>	\ominus	<code>\ominus</code>
\times	<code>\times</code>	\uplus	<code>\uplus</code>	\triangledown	<code>\triangledown</code>	\otimes	<code>\otimes</code>
\div	<code>\div</code>	\sqcap	<code>\sqcap</code>	\triangleleft	<code>\triangleleft</code>	\oslash	<code>\oslash</code>
$*$	<code>\ast</code>	\sqcup	<code>\sqcup</code>	\triangleright	<code>\triangleright</code>	\odot	<code>\odot</code>
\star	<code>\star</code>	\vee	<code>\vee</code>	\triangleleft^+	<code>\lhd^+</code>	\circ	<code>\bigcirc</code>
\circ	<code>\circ</code>	\wedge	<code>\wedge</code>	\triangleright^+	<code>\rhd^+</code>	\dagger	<code>\dagger</code>
\bullet	<code>\bullet</code>	\setminus	<code>\setminus</code>	\triangleleft^+	<code>\unlhd^+</code>	\ddagger	<code>\ddagger</code>
\cdot	<code>\cdot</code>	\wr	<code>\wr</code>	\triangleright^+	<code>\unrhd^+</code>	\amalg	<code>\amalg</code>

Other symbols

\aleph	<code>\aleph</code>	\prime	<code>\prime</code>	\forall	<code>\forall</code>	∞	<code>\infty</code>
\hbar	<code>\hbar</code>	\emptyset	<code>\emptyset</code>	\exists	<code>\exists</code>	\square	<code>\Box</code>
\imath	<code>\imath</code>	∇	<code>\nabla</code>	\neg	<code>\neg</code>	\diamond	<code>\Diamond</code>
\jmath	<code>\jmath</code>	\surd	<code>\surd</code>	\flat	<code>\flat</code>	\triangle	<code>\triangle</code>
ℓ	<code>\ell</code>	\top	<code>\top</code>	\natural	<code>\natural</code>	\clubsuit	<code>\clubsuit</code>
\wp	<code>\wp</code>	\perp	<code>\bot</code>	\sharp	<code>\sharp</code>	\diamond	<code>\diamondsuit</code>
\Re	<code>\Re</code>	\parallel	<code>\parallel</code>	\backslash	<code>\backslash</code>	\heartsuit	<code>\heartsuit</code>
\Im	<code>\Im</code>	\angle	<code>\angle</code>	∂	<code>\partial</code>	\spadesuit	<code>\spadesuit</code>
\mho	<code>\mho</code>						

Scalable symbols

Σ	Σ	<code>\sum</code>	\cap	\bigcap	\odot	\bigodot	<code>\bigodot</code>
\prod	\prod	<code>\prod</code>	\cup	\bigcup	\otimes	\bigotimes	<code>\bigotimes</code>
\coprod	\coprod	<code>\coprod</code>	\sqcup	\bigsqcup	\oplus	\bigoplus	<code>\bigoplus</code>
\int	\int	<code>\int</code>	\vee	\bigvee	\uplus	\biguplus	<code>\biguplus</code>
\oint	\oint	<code>\oint</code>	\wedge	\bigwedge			

Math functions

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

Big delimiters

<code>(</code>	<code>(</code>	<code>)</code>	<code>)</code>	\uparrow	<code>\uparrow</code>	<code>/</code>	<code>/</code>
<code>[</code>	<code>[</code>	<code>]</code>	<code>]</code>	\downarrow	<code>\downarrow</code>	<code>\</code>	<code>\backslash</code>
<code>{</code>	<code>\{</code>	<code>}</code>	<code>\}</code>	\updownarrow	<code>\updownarrow</code>	<code> </code>	<code> </code>
<code>⌊</code>	<code>\lfloor</code>	<code>⌋</code>	<code>\rfloor</code>	\Uparrow	<code>\Uparrow</code>	<code> </code>	<code>\ </code>
<code>⌈</code>	<code>\lceil</code>	<code>⌉</code>	<code>\rceil</code>	\Downarrow	<code>\Downarrow</code>		
<code>⟨</code>	<code>\langle</code>	<code>⟩</code>	<code>\rangle</code>	\Updownarrow	<code>\Updownarrow</code>		

Accents

<code>\hat{a}</code>	<code>\hat{a}</code>	<code>\acute{a}</code>	<code>\acute{a}</code>	<code>\bar{a}</code>	<code>\bar{a}</code>	<code>\dot{a}</code>	<code>\dot{a}</code>
<code>\check{a}</code>	<code>\check{a}</code>	<code>\grave{a}</code>	<code>\grave{a}</code>	<code>\vec{a}</code>	<code>\vec{a}</code>	<code>\ddot{a}</code>	<code>\ddot{a}</code>
<code>\breve{a}</code>	<code>\breve{a}</code>	<code>\tilde{a}</code>	<code>\tilde{a}</code>				

Typefaces in math environments

<code>\mathnormal{text}</code>	implicit math italic
<code>\mathrm{text}</code>	typeface Roman
<code>\mathbf{text}</code>	bold typeface Roman
<code>\mathsf{text}</code>	sans serif
<code>\mathit{text}</code>	math italic
<code>\mathtt{text}</code>	typewriter
<code>\mathcal{text}</code>	calligraphic font (only English capital letters).

How to install and use a working distribution

To get started with $X_{\text{T}}\text{L}\text{A}\text{T}\text{E}\text{X}$ (or $\text{L}\text{A}\text{T}\text{E}\text{X}$ and other TEX -like systems), there are several distributions available for free. The TEX Live distribution has been carefully maintained and updated every year for many years as the most widespread and most easily available. The original idea of this distribution was its ease of use even without any installation; by simply inserting the media (DVD) into the drive it is possible to start working. However, this is an atypical use case, for serious work it is advisable to install the distribution on your own computer.

The distribution is available from the Internet <https://tug.org/texlive/acquire-netinstall.html>, where you can also find all other information about this distribution, installation process, documentation, etc.

The entire installation can be started by downloading the file `install-tl-windows.exe` (for Windows-type systems) or `install-tl-unx.tar.gz` for other platforms.

It is a relatively small archive, unpacking which creates the directory `install-tl-*` and there is the installation script `install-tl-windows.bat` (or similar for other platforms). It needs to be run. After setting several installation parameters (distribution target directory, etc.), the installation starts, downloads all the necessary files from the Internet (several gigabytes) and finally generates the necessary formats and sets other elements.

It is advisable to supplement the distribution with a suitable editor for editing and updating the source text. There are a large number of such editors, and each user can choose the one that suits him best. For example, the Open Source editor TEX studio (available at texstudio.org) can be chosen as one of the widely used ones.

In addition to installing the system on your own computer, there is also the possibility of using web applications. One of them is the Overleaf application (overleaf.com). It is available for free for general use, advanced functions and project sharing are charged. The TEX onWeb application was also originally created for teaching purposes at the Faculty of Business Economics of the Mendel University in Brno (tex.mendelu.cz). It can be used completely free of charge for any purpose.

B.1 User Support

Users of TEX and all related systems can find many components (packages and supporting software) on the CTAN repository – Comprehensive TeX Archive Network (ctan.org, which is mirrored worldwide – over 100 servers).

Users of \TeX all over the world gather in groups – TUG (\TeX Users Groups). The main group is the American TUG and there are 27 other user groups around the world. C \S TUG – Czech and Slovak \TeX Users Group operates in the Czech and Slovak Republic (details on the website www.cstug.cz). Membership in a TUG supports the development of the system and its use in all areas. Some groups also publish specialist magazines dealing with the development and use of \TeX -based systems. TUGBoat (TUG magazine) is well-known and spread worldwide, and the C \S TUG Bulletin is published in the Czech and Slovak Republics. Members receive the magazine in printed form, for other people electronic archive versions are also available (usually with some delay).

In addition to user support within the \TeX groups, it is also possible to use the exchange of good practice at <https://tex.stackexchange.com/>.

For inspiration – how this publication is typeset

In the following abbreviated view of the document class `ioscs.cls` you can see how some elements of this book were implemented. Some standard commands have been redefined, and specific definitions have been added to them according to the needs of this document.

```
%%
%% This is file `ioscs.cls',
%%
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{ioscs}[2020/01/11 XeLaTeX document class
                    for Erasmus+ documents]

\LoadClass[twoside]{report}      % default font size: 10 pt
\usepackage[b5paper]{geometry}  % paper format
\usepackage{unicode-math}
\usepackage{xltextra, polyglossia}
\setdefaultlanguage{english}
\usepackage[usenames]{xcolor}
\usepackage{tabularx}
\usepackage{colortbl}
\usepackage{graphicx}
\usepackage{fancyvrb}
\usepackage{longtable}
\usepackage{multicol}
\usepackage{url}

% font for books:

\defaultfontfeatures{Mapping=tex-text}

\setmainfont{Libertinus Serif}
\setsansfont{Libertinus Sans}
\setmathfont{Libertinus Math}

%%%%%%%%%%%% colors

\definecolor{myblue}{rgb}{0.3,0.5,0.74} % color for emphasize
```

%% Microtypography

```
\def\uv#1{"#1"} % quotes: english
%\def\uv#1{„#1"} % quotes: czech, slovak
\def\bsl{\char92\relax} % backslash
\def\lbr{\char123} % curly brackets
\def\rbr{\char125}
\def\striska{$\hat{\rule{0.5em}{0pt}}}$}
\def\, {\penalty10000\hskip0.25em} % solid 1/4 em space
\def\; {\penalty10000\hskip0.16667em} % solid 1/6 em space
\def\spoj{\discretionary{-}{-}{-}} % czech hyphen
```

%% Sectioning

% redefined chapter heads

```
\def\chapter {%
  \if@openright \cleardoublepage \else \clearpage \fi
  \thispagestyle {empty}%
  \global \@topnum \z@
  \@afterindentfalse
  \secdef \@chapter \@schapter}
```

```
\renewcommand\thechapter{\arabic{chapter}}
```

```
\def\@makechapterhead #1{%
  {\parindent \z@ \raggedright \Large \bfseries
    \interlinepenalty\@M
    \textsf{\color{myblue}}\@chapapp
      \enspace\thechapter\hfill\thepage}
  \hbox{\rule[20pt]{\textwidth}{0.8pt}}
  \huge \bfseries
  \begin{flushright}
  \strut{\color{myblue}#1}
  \end{flushright}
  \nobreak
  \vskip 25\p@ plus 10\p@ minus 4\p@
  \@topnum=0
  }%
}
```

```
\def\@makeschapterhead#1{%
  {%
  \markboth{}{}%
  \parindent \z@ \raggedright \normalfont
  \interlinepenalty\@M
  \textsf{\color{myblue}}\mbox{}\hfill\thepage}
```

```

    \hbox{\rule[20pt]{\textwidth}{0.8pt}}
    \huge \bfseries
    \begin{flushright}
    \strut{\color{myblue}#1}
    \end{flushright}
    \par\normalsize%
    \nobreak
    \vskip 40\p@ plus 10\p@ minus 4\p@
    \@topnum=0
  }%
}

% redefined sectioning from the latex.ltx file:
\def\section{\@startsection {section}{1}{\z@}%
  {-3.5ex \@plus -1ex \@minus -.2ex}%
  {2.3ex \@plus .2ex}%
  {\normalsize\Large\bfseries\color{myblue}\raggedright}}
\def\subsection{\@startsection{subsection}{2}{\z@}%
  {-3.25ex\@plus -1ex \@minus -.2ex}%
  {1.5ex \@plus .2ex}%
  {\normalfont\large\bfseries\color{myblue}\raggedright}}
\def\subsubsection{\@startsection{subsubsection}{3}{\z@}%
  {-3.25ex\@plus -1ex \@minus -.2ex}%
  {1.5ex \@plus .2ex}%
  {\normalfont\normalsize\bfseries\color{myblue}\raggedright}}
\def\paragraph{\@startsection{paragraph}{4}{\z@}%
  {3.25ex \@plus 1ex \@minus .2ex}%
  {-1em}%
  {\normalfont\normalsize\bfseries\color{myblue}\raggedright}}
\def\subparagraph{\@startsection{subparagraph}{5}{\parindent}%
  {3.25ex \@plus 1ex \@minus .2ex}%
  {-1em}%
  {\normalfont\normalsize\bfseries\color{myblue}\raggedright}}

%%%%%%%%%%%%%% Page concept

\textwidth 126mm \hoffset 0mm
\textheight 200mm \voffset -10mm
\evensidemargin 0mm
\oddsidemargin 0mm

% paragraph typesetting
\parindent 2em
\parskip 0pt
\raggedbottom
\tolerance 2000
\clubpenalty 10000

```

```

\pagenumbering{arabic}

\def\folio{\fontsize{12pt}{14pt}\selectfont\bfseries
\ssfamily\color{myblue}}

% redefined plain page style:
\def\ps@plain{\def\@oddhead{}\def\@evenhead{}%
\def\@oddfoot{\parbox{\textwidth}{\raggedleft\folio\thepage}}
\def\@evenfoot{\parbox{\textwidth}{\folio\thepage}}%
}

%%%%%%%%%%%%% emphasizing

\def\strong#1{\textbf{#1}}
\def\vpar#1{{\color{myblue}\itshape\mdseries\rmfamily#1/}}

\let\silvyznacnt\bfseries % czech equivalents
\let\vyznacnt\itshape
\let\vyznac\em

\def\newterm#1{{\bfseries\color{myblue}#1}}

%%%%%%%%%%%%% figures

\def\mycaption#1{\refstepcounter{figure}
\parbox{\textwidth}{\raggedright
\small\itshape \figurename{} \thefigure: #1}%
\addcontentsline{lof}{section}{\thefigure: #1}}
\def\ourfigure#1#2{\begin{figure}[htbp]\centering
\includegraphics{#1.pdf}\par\bigskip
\mycaption{#2}\label{#1}\end{figure}}
\def\ourfigurefit#1#2{\begin{figure}[htbp]\centering
\includegraphics[width=\linewidth]{#1.pdf}\par
\bigskip\mycaption{#2}\label{#1}\end{figure}}
\def\ourfigureex#1#2#3#4{\begin{figure}[htbp]\centering
\includegraphics[#4]{#1}\par\bigskip\mycaption{#2}%
\label{#3}\end{figure}}

%%%%%%%%%%%%% tables, tabulars

\def\psframe#1#2{{\unitlength\textwidth\begin{picture}(1,0)
\put(0,0){\color{#1}\rule[-3mm]{\textwidth}{13mm}}
\put(0.5,0){\makebox(0,0)[bc]{\vrule depth 8mm width 0pt
\parbox[t]{\textwidth}{\centering #2}}}
\end{picture}}}

```



```

\def\pole#1#2{\begin{tabular}{@{}#1@{}}#2\end{tabular}}
\def\polet#1#2{\begin{tabular}[t]{@{}#1@{}}#2\end{tabular}}

\def\tabpozn#1#2{\multicolumn{#1}{@{}l}{\formpoznfsnt #2}}

\def\horizcara{\par\hrule\relax}

\def\tabname{Tab.}
\newdimen\popis
\long\def\caption#1{\refstepcounter{table}%
  \addcontentsline{lot}{section}{\thetable: #1}%
  \setbox0=\hbox{\tabname\ \thetable:\ }
  \popis=\linewidth \advance\popis by -\wd0
  \unhbox0\parbox[t]{\popis}{\raggedright\vyznacfn#1}
  \par\medskip\nobreak}

\newlength{\aktsirkatab}
\newcounter{@tmpsloupce}
\def\preparetab#1{\aktsirkatab=\linewidth
  \setcounter{@tmpsloupce}{#1}%
  \advance\aktsirkatab by -#1\tabcolsep
  \advance\aktsirkatab by -#1\tabcolsep}

\def\ourtable#1{\begin{table}[htb]\centering
  \ifempty#1\else\caption{#1}\fi}

\def\endtab{\end{table}}

```

Open Source Tools for Text Processing
Study text

Author: doc. Ing. Jiří Rybička, Dr.
Mendel University in Brno

Publisher: Mendel University in Brno, Zemědělská 1, 613 00 Brno, Czech Republic

Graphic editing and typesetting: Jiří Rybička

Year of publishing: 2022

First edition

Number of pages: 106

ISBN 978-80-7509-880-1 (online ; pdf)

DOI <https://doi.org/10.11118/978-80-7509-880-1>