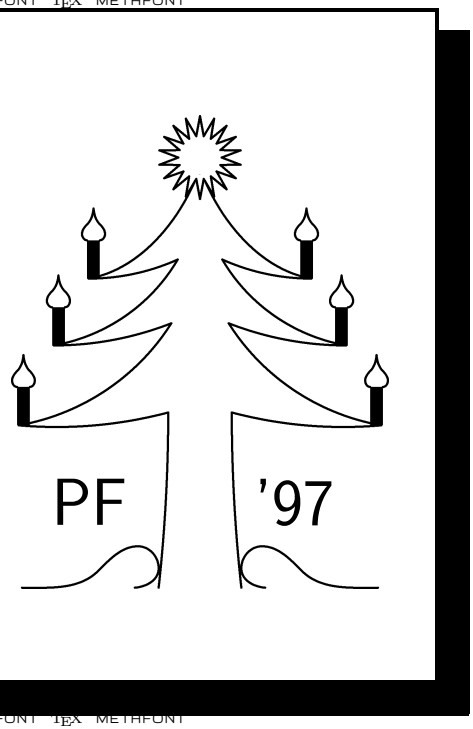
TEX METAFONT



4

ZPRAVODAJ 96

ČESKOSLOVENSKÉHO SDRUŽENÍ UŽIVATELŮ TEXU

OBSAH

Zpravodaj Československého sdružení uživatelů TeXu je vydáván v tištěné podobě a distribuován zdarma členům sdružení. Po uplynutí dvanácti měsíců od tištěného vydání je poskytován v elektronické podobě (PDF) ve veřejně přístupných archívech.

Starší čísla Zpravodaje budou zveřejněna v elektronické podobě v nejbližší době. Vyzýváme autory, kteří s tímto způsobem zveřejnění svých článků nesouhlasí, aby o tom uvědomili redakci Zpravodaje.

Místo úvodníku . . .

Zdeněk Wagner

V okamžiku, kdy vaše oči začnou vnímat tyto řádky, bude se chýlit ke konci další rok činnosti Československého sdružení uživatelů TEXu, nebo možná nový rok bude začínat. Při troše štěstí se toto číslo Zpravodaje může objevit symbolicky pod vánočním stromečkem, který jistě bude vypadat mnohem lépe než ten, jenž ilustruje obálku.

Nebudeme se zde pouštět do bilancí toho, co se podařilo, a co se nepovedlo. Není to v kompetenci redakce a nakonec vše nejlépe víte vy, členové sdružení. Vy sami víte, zda jste či nejste s činností sdružení spokojeni, vy sami víte, co byste si nejraději přečetli na stránkách dalších čísel Zpravodaje.

Na tomto místě se sluší poděkovat všem, kteří k vydání čtyř letošních čísel přispěli. V první řadě je to Karel Horák, který poskytl svá makra a díky tomu se vzhled Zpravodaje příliš nezměnil. Dále je to nepochybně Petr Olšák, z jehož pera pochází významná část příspěvků. Stejně tak je nutno poděkovat tiskařům, jejichž jména ani neznám, jakož i všem ostatním autorům článků, které se letos na stránkách Zpravodaje objevily. Výčet asi nemůže být úplný a věřím, že mě omluví všichni, jejichž jména se tu nevyskytují.

Nabízí se otázka, co by měl Zpravodaj poskytnout v následujícím roce. Na jedné výborové schůzi padl návrh, že jednotlivá čísla by měla být monotematická. S takovým názorem redakce souhlasí, nicméně ve skromném archivu není dostatek příspěvků, aby se dalo vytvořit důstojné číslo týkající se nějakého problému.

Jaký tedy Zpravodaj bude? Bude takový, jaký si jej uděláte vy, členové sdružení. Jistě máte nějaké zkušenosti, o něž se můžete s ostatními podělit. Máte i vlastní názory opřené o praxi v používání TEXu, které by třeba pomohly těm, kteří se v počítačové typografii teprve začínají orientovat. Někteří z vás vytvořili zajímavé pomocné programy nebo užitečná makra. Zpravodaj poskytuje vhodný prostor pro jejich prezentaci. Stačí napsat článek a odeslat jej na adresu uvedenou v tiráži.

Není řečeno, že Zpravodaj má vycházet striktně čtyřikrát ročně. Lze vydat šest i více čísel, pokud ovšem bude v redakci dostatek materiálu.

Pro čtyři čísla příštího roku jsou uzávěrky stanoveny na dny 14. února, 23. května, 18. července a 31. října.

V okamžiku, kdy vaše oči vnímají tyto řádky, chýlí se možná starý rok ke konci, možná nový rok začíná. Redakce vám přeje, aby nový rok byl pro vás po všech stránkách úspěšný a sobě přeje, aby vás i další čísla Zpravodaje uspokojila. Těší se též na spolupráci s již zavedenými i s novými autory, těší se na články, které jistě přinesou všem řadu nových informací.

Pour féliciter '97.

Zdeněk Wagner
bulletin@cstug.cz
zpravodaj@cstug.cz

Výbor Československého sdružení uživatelů TEXu rozhodl, že ocení originální program přispívající k rozšíření funkčních možností systému TEX. Tímto programem je

TEX2PDF

Článek o tomto programu jste si mohli přečíst ve Zpravodaji č. 2/96 na str. 69–85. Autorem programu je

Han The Thanh

Výbor udělil autorovi programu finanční odměnu ve výši 5 000 Kč.

CSTUG, Charles University, Prague, March 1996 Questions and Answers with Prof. Donald E. Knuth

Dr. Karel Horák:
[Introductory remarks in Czech, then English.]

I'm very glad to have such a happy occasion to introduce you, Professor Knuth, to our audience, who are mostly members of CSTUG, the Czech/Slovak TEX User Group, but also some academicians from Prague because this session is organized by CSTUG and the Mathematics Faculty of Charles University. We are very happy to have you here, and I would be happy, on behalf of Charles University, to give you a special medal. [wide applause]

DEK: [surprised] Thank you very much.

Prof. Ivan Netuka: Professor Knuth, dear colleagues, dear friends, ladies and gentlemen. I feel really very much honored having the opportunity to greet Professor Donald Knuth, as well as most of you here sitting in this guildhall, on behalf of the Dean of the Faculty of Mathematics and Physics of Charles University, Professor Bedřich Sedlák.

As far as I know, Professor Knuth has come to Prague for the first time. Despite this fact, he has been known here, not only among *all* mathematicians, *all* computer scientists, but also many physicists, and even to people having nothing to do with our subjects. People here are fully aware of the significance of Donald Knuth's [...] treatise, *The Art of Computer Programming*. Many of us have had the opportunity to be pleased by reading the charming booklet devoted to *Surreal Numbers*. We know—and here I am going to fall [stumbled]—that Donald Knuth's favorite way to describe computer science is to say that it is the study of algorithms. We share his opinion that the study of algorithms has opened up a fertile vein of interesting new mathematical problems and that it provides a stimulus for many areas of mathematics which have been suffering from a lack of new ideas.

My personal experience—the personal experience of a mathematician—says that, for every mathematician, there exists a personality who [has] brought an extraordinarily great service to his field. Here we have a rare case where, in that statement, the order of the quantifiers may be reversed, maybe: There exists a personality who [has] brought a great service, an extraordinarily great service, to every mathematician. Here is my one-line proof: Donald Knuth—TEX.

Professor Knuth, in acknowledgement of your achievements in computer science, in mathematics, as well as in computerized typography, which has given the whole of the community an excellent tool for presenting scientific results, the Faculty of Mathematics and Physics of Charles University [has] decided that you be awarded the Faculty's Memorial Medal. I am happy to make that presentation now. [wide prolonged applause]

Obrázek 1: The Seal of Charles University

DEK: Well, this is a quite beautiful medal; I hope you can come and look at it. "Universitas Carolina Pragensis"—so we all speak Latin; maybe I should speak Latin today. [laughter]

I don't know much about the Czech language, but I've tried to learn some of it. On many doors this week I see the word "Sem". [laughter] And then as I came up to this lecture hall today, there were many other signs that said "TEX". [laughter] So I thought we could have an especially powerful version of TEX [writes 'SemTEX' on the blackboard; more laughter] but perhaps it's dangerous; I don't know....

This morning I have no prepared lecture, but I want to say just what you want to hear, so I want to answer your questions. This is a tradition that I maintained in California: The very last session of every class that I taught at Stanford was devoted to questions and answers. I told the students they didn't have to come to that class if they didn't want to, but if they came I would answer any question that they hoped to have answered when they signed up for the class. I actually borrowed this tradition from Professor [Richard] Feynmann at Caltech. And I decided I would do it in my classes, too; it's a wonderful idea that I recommend to all professors—to have open-ended question and answer sessions.

I've recently made some home pages on the World Wide Web that you can get via

`http://www-cs-faculty.stanford.edu/~knuth`

and there on those pages I have the answers to all frequently asked questions. But today, you can ask me the *un*frequently asked questions. [laughter] By the way, I'll tell you one more joke and then we'll get started. Do you know what the home page is of OJ—O.J. Simpson—in the United States? It's "http colon slash slash slash backslash slash escape". [laughter]

Now, please ask me questions. [pause]

Well, if there are no further questions, ... [laughter]. You may ask in Czech, and then someone will translate.

?: Maybe a question to start [with]. I learned TEX carefully, and I had a problem when someone asked me to take the integral with tilde accent. I found that maybe there isn't one with TEX because you can't specify an italic correction to boxes.

DEK: The italic correction is ... With each character there's a limited amount of information that goes in the data structure for each character, and so we have [drawing on blackboard] the height, the depth, the width, and the italic correction. But those are the only numbers that are allowed, and in mathematics mode, the italic correction is used in a

different way from outside of mathematics. In mathematics mode, the italic correction is actually used for subscripts; it's the amount by which you would bring the subscript to the left—otherwise, it would typeset "P sub n" (P_n) like this: P_n.

The italic correction on the integral sign might even be another case because the large operators use the italic correction to cover the spacing between the lower limit and the upper limit. Anyway, there's only one number in there. If you want a special construction that demands many more numbers, the only way I know is to make a special macro for that. I would carry the information somewhere up in the TEX level, not in the inside, not with the character. You would have to build a structure that has this information in it. I don't know how general a solution you need, but certainly if you said the ... I can't even remember the name now ... my goodness, how do you get the ... like the same mechanism by which someone would take an equal sign and then put something over it, like this. It's defined in plain TEX by a macro ...

?: It's something like `\mathord` and upper limits ... [he means `\buildrel`]

DEK: I would build it up out of the primitives, but if you had different integral signs, you would probably have to allow the person who specified the font to ...

?: I have a solution, but it is not a TEX solution: I used METAFONT to produce special characters, which have the [...]

DEK: Yes, using METAFONT would be the ideal way to get the correct artistic effect, but then everyone else has to get your METAFONT code and compile your font. Just by a combination of boxes and glue, you should be able to position the characters that you have. You could just make a `\vbox` [drawing on blackboard] or a `\vcenter` of something or other, and then you build the `\hbox` of ... with a kern and then a tilde or so on. Otherwise, I don't know any simple way of doing exactly that balancing because it's complicated by the visual proportions of the spacing with integral signs—it gets really complicated to handle *all* cases.

My general philosophy with TEX was to try to have a system that covers 99% of all cases easily [laughter]; and I knew there would always be a residual number. But I felt that this residual would only be needed by the people who really care about their papers, and then if they're

only spending 1% of the time on this, then they would enjoy feeling that they had contributed something special by adding their little signature, their special character to it. So, I didn't try to do everything automatically. I still believe that it's worthwhile thinking about how to do more automatically, but I don't believe you ever get all the way there.

Dr. Karel Horák: I would be very interested in your way of thinking—when you started thinking about making TEX and the typesetting system—when you realized that you also needed to produce some letters, to have not only TEX but also METAFONT. Because—I don't know too much about all types [typefaces] of digital typography—but I think there weren't very many types which you *could* use with TEX, so probably you started thinking about METAFONT, about something like that, from the first?

DEK: Exactly.

I have to erase this beautiful calligraphy [laughter]. It's too late now; well, whoever did it can do it again later, but I need the board. It's gorgeous, although this should really be a different "A". [laughter]

Let's go back to April 1977. [writes on board] I sat down at a computer terminal and started writing a memorandum to myself about what I thought would be a good language for typesetting. And in May 1977, I began working on fonts. This was going to be my sabbatical year, where I would do no teaching through the end of 1977, and the beginning of 1978. I thought that I would write a typesetting system just for myself and my secretary. [laughter] I had no idea that I would ever be seeing TEX on, for example, the tram signs in Brno [laughter] or by the churches of the city, and so on. It was just for my own purposes, and I had one year to do it. And I thought it would be easy. So, in May of 1977, I went to Xerox PARC, the place where the ideas of mouses and windows and interfaces and so on were being worked on, and I knew that they were playing with splines for letterforms. I saw Butler Lampson at a computer terminal, and he was adjusting splines around the edges of letters that he had magnified; so I thought, "good, I'll make an arrangement to work at Xerox PARC during my sabbatical year, and use their cameras and make the type."

I knew from the beginning that I wanted the type to be captured in a purely mathematical form; I wanted to have something that would adapt to technology as it kept changing, so that I would have a permanent

mathematical description of the letters. Unfortunately, Xerox said, "Yes, you're welcome to use our equipment, but then we will own the designs, they will be the property of Xerox." I didn't want any of this work to be proprietary; I didn't want people to have to pay to use it. ... A mathematical formula is just numbers—why shouldn't everybody own these numbers?

So instead, I worked only at Stanford, at the Artificial Intelligence Laboratory, with the very primitive equipment there. We did have television cameras, and my publisher, Addison-Wesley, was very helpful—they sent me the original press-printed proofs of my book, from which *The Art of Computer Programming* had been made. The process in the 60s that I wanted to emulate was interesting: They would first print with metal type, Monotype, onto good paper, one copy. They made one copy with the metal, then they photographed that copy and printed from the photograph. They gave me that original copy from which they had made the original photographs. So I could try putting the TV camera on that, and go from the TV camera to a computer screen to copy the letters. At that time, we could connect our display terminals to television and movies on television; people were looking at the titles of movies, and capturing the frames from the movies and then making type. They would keep waiting for more episodes of *Star Trek* or something so that we would have the whole alphabet; eventually we would get a title with the letter "x" in it. That's how we were trying to get type by means of television at the time.

I thought it would be easy, but immediately I noticed that if I turned the brightness control a very little bit, the letters would get much thicker. There was a tremendous variation, so that what I would see on my TV screen had absolutely no consistency between a letter that I did on Monday and a letter that I did on Tuesday, the following day. One letter would be fat and one letter would be thin, but it would be the same letter because the brightness sensitivity was extremely crude. This is still true now: If you look at a scanner and you change the threshold between black and white, a small change in the threshold changes the character of the letter drastically. So I couldn't use TV.

For the next attempt, my wife made photographs of the pages and then we took our projector at home and projected them down a long hallway. On the wall I would try to copy what the letters were. But at that point I realized that the people who had designed these typefaces

actually had ideas in their mind when they were doing the design. There was some logic behind the letters. For example, you have the letter 'm', you have the letter 'n', you have an 'i' and an 'l', and I noticed that the 'm' was 15 units, the 'n' was 10 units, and the 'i' was 5 units. Aha! A pattern! The 'l' was 5 units, the 'f' was 5 units, the 'fi' ligature was 10 units. So, if you cut off the tops of these letters, you would see an exact rhythm of 5 units between stems. Great—there were regularities in the design! That's when it occurred to me that maybe I shouldn't just try to copy the letterforms, but I should somehow try to capture the intelligence, the logic, behind those letterforms. And then I could do my bold font with the same logic as the regular font.

The truth therefore is that in May 1977 I didn't know what to do about fonts; June 1977 is when I started to have the idea of METAFONT.

I spent the summer of 1977 in China, and I left my students in California; I told them to implement TEX while I was gone. [laughter] I thought it would be very easy; I would come home and they would have TEX working, and then I could do the fonts. But when I got back, I realized that I had given them an impossible task. They actually had gotten enough of TEX running to typeset one character on one page, and it was a heroic achievement, because my specifications were very vague. I thought the specifications were precise, but nobody understands how imprecise a specification is until they try to explain it to a computer. And write the program.

When I was not in China—in June, the first part of July, and September, October, November—I spent most of my time making fonts. And I had to, because there was no existing way to get a font that would be the same on different equipment. Plenty of good fonts existed, but they were designed specifically for each manufacturer's device. There was no font that would go to two devices. And the people at Xerox PARC—primarily John Warnock—were still developing their ideas; they eventually founded Adobe Systems about 1980 or so. Now, with the help of many great designers, they have many beautiful fonts. But that came later, about two or three years after I had an urgent need for device-independent type.

My lecture to the American Math Society was scheduled for January 1978. The transcript of the lecture that I gave, the Gibbs Lecture to the Society, shows the work that I did with fonts in 1977. It was a much longer task than I ever believed possible. I thought it would be simple

to make something that looked good—it was maybe six years before I had anything that I really was satisfied with.

So, the first big ideas were to get fonts that would be machine independent and work on many different computers, including future ones that had not been invented, by having everything defined in mathematics. The second idea was to try to record the intelligence of the design. I was not simply copying a shape, I also would specify that if part of the shape changes, the other should change in a logical way. My goal was to understand the designer's intention, and not just copy the outcome of the intention.

Well, I didn't have TEX running until May of 1978—I didn't have TEX—I drew the fonts first. For the article, "Mathematical Typography", my talk to the American Math Society,[1] I made individual letters about 4cm high and I pasted each one on a big sheet of paper and took a photograph of that.

That's a long answer. I hope I answered the question.

Dr. Karel Horák: I have another question about this system; it is, when you started to learn typography, you had some knowledge before, or you started in the process, learning more and more? Because my experience with *The TEXbook*, and [that of] others also, is that there is very much about typography. You can learn a lot about typography, much more than some people who are doing typesetting on the professional level, using those windows mouse systems. They never can learn from the books which are supplied with those systems.

DEK: Thank you. So, what was my background before 1977? When I was in secondary school—like gymnasium—I had a part-time job setting type (so-called) on what was known as a mimeograph machine. I'm not sure what would be the equivalent here. On a mimeograph you had a sort of blue gelatinous material. The typewriter typed into it and it made a hole. I would also use a light table, and special pens, and try to make music or designs on the mimeograph stencil. I had a summer job where I would type, and then I would use my stylus to inscribe pictures on the gel. So I knew a little bit about typography. This was not fine printing, of course; it was very amateurish, but at least it gave me some idea that there was a process of printing that I could understand. After

[1] *Bull. (N.S.) Amer. Math. Soc.* **1** (1979), pages 337–372; republished in *TEX and METAFONT: New Directions in Typesetting*, Bedford, MA: Digital Press, 1979.

making the stencils, I would run the machine, and cut the paper, and so on. I was doing this as a student.

Later, my father had a printing press in the basement of our house, and he did work for the schools of Milwaukee; this was to save money from going to the professional places. He would work for some architects that were friends of ours, to make their specification documents. Also in the schools, there would be a program for a concert, or graduation ceremony, something like that, printing tickets for football games ... he would do this in our basement. He started with a mimeograph machine, then he upgraded to something called a VariTyper, which was marvelous, because it had proportional spacing—some letters were wider than others; the fonts were terrible, but we had this machine, and I learned how to use it.

Still later, I started writing books, *The Art of Computer Programming*. So, by 1977, I had been proofreading thousands of pages of galley proofs. I certainly was looking at type. And you might say I was getting ink in my blood.

But I also knew that engineers often make the mistake of not looking at the traditions of the past. They think that they'll start everything over from scratch, and I knew that that was terrible. So actually, right during April and May of 1977, when I was thinking about starting my sabbatical year of typesetting, I took a trip with the Stanford Library Associates, a group of book lovers from Stanford. We visited places in Sacramento, California, where people had special printing presses. We stopped at a typographic museum, which had a page from a Gutenberg Bible, and so on. Everywhere we went on this tour, I looked intently at all the letters that I saw. And I saw people's collections of what they felt was the finest printing.

At Stanford Library there is a wonderful collection of typographic materials donated by a man named Gunst, who spent a lifetime collecting fine printing. As soon as I got back from the library trip, I knew about the Gunst collection, so I spent May and June reading the works of Goudy and Zapf and everything I could find, back through history. First of all, it was fascinating, it was wonderful, but I also wanted to make sure that I could capture as well as possible the knowledge of past generations in computer form.

The general idea I had at that time was the following. At the beginning, when I was young, we had computers that could deal only with

numbers. Then we had computers that knew about numbers *and* capital letters, uppercase letters. So this greatly increased our ability to express ourselves. Even in Volume 1 of *The Art of Computer Programming* when I designed my `MIX` computer, I never expected that computers could do lowercase letters. [laughter] The Pascal language was developed approximately 1968, 1969; Pascal originally used only uppercase letters, and parentheses, commas, digits, altogether 64 characters.

Next, in the early 1970s, we had lowercase letters as well, and computers could make documents that looked almost like a typewriter. And then along came software like the *eqn* system of UNIX, which would make documents that approached printing. You probably know that *troff* and the *eqn* system for mathematics were developed at Bell Labs. This was an extension of a program that began at MIT in 1959 or 1960, and it developed through a sequence of about five levels of improvement, finally to *eqn* in 1975.

So I knew that it was possible, all of a sudden, to get better and better documents from computers, looking almost like real books. When contemplating TEX I said, "Oh! Now it's time to go all the way. Let's not try to *approach* the best books, let's march all the way to the end—let's do it!" So my goal was to have a system that would make the best books that had ever been made, except, of course, when handmade additions of gold leaf and such things are added. [laughter] Why not? It was time to seek the standard for the solution to all the problems, to obtain the very best, and not just to approach better and better the real thing. That's why I read all the other works that I could, so that I would not miss any of the ideas. While reading every book I could find in the Gunst collection, to see what they could tell me about typesetting and about letterforms, I tried to say, "Well, how does that apply, how could I teach that to a computer?" Of course, I didn't succeed in everything, but I tried to find the powerful primitives that would support most of the ideas that have grown up over hundreds of years.

Now, of course, we have many more years of experience, so we can see how it is possible to go through even many more subtle refinements that I couldn't possibly have foreseen in 1980. Well, my project took more than one year, and I had more than one user at the end. The subsequent evolution is described in my paper called "The Errors

of TEX", and the complete story after 1978 is told in that paper.[2]

In 1980, I was fortunate to meet many of the world leaders in typography. They could teach me, could fill in many of the gaps in my knowledge. Artisans and craftsmen usually don't write down what they know. They just do it. And so you can't find everything in books; I had to learn from a different kind of people. And with respect to type, the interesting thing is that there were two levels: There was the type *designer*, who would draw, and then there was the *punchcutter*, who would cut the punches. And the type designer would sometimes write a book, but the punchcutter would not write a book. I learned about optical illusions—what our eye thinks is there is not what's really on the page. And so the punchcutter would not actually follow the drawings perfectly, but the punchcutter would distort the drawings in such a way that after the printing process was done and after you looked at the letter at the right size, what you saw was what the designer drew. But the punchcutter knew the tricks of making the right distortions.

Some of these tricks are not necessary any more on our laser printers. Some of them were only for the old kind of type. But other tricks were important, to avoid blots of ink on the page and things like that. After I had done my first work on METAFONT, I brought Richard Southall to Stanford; he had been working at Reading University with the people who essentially are the punchcutters. He gave me the extra knowledge that I needed to know. For example, when stems are supposed to look exactly the same, some of them are a little bit thinner, like the inside of a 'p'—you don't want it to be quite as thick, you want it to be a little thinner; then, after you have the rest of the letter there, the lightened stem will look like it was correct. Richard taught me that kind of requirement. I learned similar things from Matthew Carter, Hermann Zapf, Chuck Bigelow, Gerard Unger, and others.

But we had very primitive equipment in those days, so that the fonts that we could actually generate at low resolution did not look professional. They were just cheap approximations of the fine type. Stanford could not afford an expensive typesetting machine that would realize our designs at the time. Now I'm so happy that we have machines like the

[2] *Software—Practice and Experience* **19** (1989), pages 607–681. Reprinted with additional material in *Literate Programming* (CSLI Publications, Stanford, 1992, and Cambridge University Press), pages 243–339.

LaserJet 4, which make my type look the way I always wanted it to look, on an inexpensive machine.

?: Now that PostScript is becoming so widely used, do you think it is a good replacement for METAFONT—I mean, good enough? Right now, we can use TEX and PostScript ...

DEK: The question is, is PostScript a good enough replacement for METAFONT?

I believe that the available PostScript fonts are quite excellent quality, even though they don't use all of the refinements in METAFONT. They capture the artwork of top-quality designs. The multiple master fonts have only two or three parameters, while Computer Modern has more than sixty parameters; even with only two or three it's still quite good. The Myriad and Minion fonts are excellent.

I'm working now with people at Adobe, so that we can more easily substitute their multiple master fonts for the fonts of public-domain TEX documents. The goal is to make the PDF files smaller. The Acrobat system has PDF files which are much larger—they're ten times as big as `dvi` files, but if you didn't have to download the fonts, they would only be three times as large as the `dvi` files. PDF formats allow us search commands and quite good electronic documents. So I'm trying to make it easier to substitute the multiple master fonts. They still aren't quite general enough. I certainly like the quality there.

Adobe's font artists, like Carol Twombly and Robert Slimbach, are great; I was just an amateur. My designs as they now appear are good enough for me to use in my own books without embarrassment, but I wouldn't mind using the other ones. Yes, I like very much the fonts that other designers are doing.

Asking an artist to become enough of a mathematician to understand how to write a font with 60 parameters is too much. Computer scientists understand parameters, the rest of the world doesn't. Most people didn't even know the word 'parameters' until five years ago—it's still a mysterious word. To a computer person, the most natural thing when you're automating something is to try to show how you would change your program according to different specifications. But this is not a natural concept to most people. Most people like to work from a given set of specifications and then answer that design problem. They don't want to give an answer to all possible design specifications that they

might be given and explain how they would vary their solution to each specification. To a computer scientist, on the other hand, it's easy to understand this kind of correspondence between variation of parameters and variation of programs.

In the back?

Láďa Lhotka: I have a problem for you. [question about structured programming]

DEK: I was talking with Tony Hoare, who was editor of a series of books for Oxford University Press. I had a discussion with him in approximately ... 1980; I'm trying to remember the exact time, maybe 1979, yes, 1979, perhaps when I visited Newcastle? I don't recall exactly the date now. He said to me that I should publish my program for TEX.[3]

As I was writing TEX I was using for the second time in my life ideas called "structured programming", which were revolutionizing the way computer programming was done in the middle 70s. I was teaching classes and I was aware that people were using structured programming, but I hadn't written a large computer program since 1971. In 1976 I wrote my first structured program; it was fairly good sized—maybe, I don't know, 50,000 lines of code, something like that. (That's another story I can tell you about sometime.) This gave me some experience with writing a program that was fairly easy to read. Then when I started writing TEX in this period (I began the implementation of TEX in October of 1977, and I finished it in May 78), it was consciously done with structured programming ideas.

Professor Hoare was looking for examples of fairly good-sized programs that people could read. Well, this was frightening. This was a very scary thing, for a professor of computer science to show someone a large program. At best, a professor might publish very small routines as examples of how to write a program. And we could polish those until ... well, every example in the literature about such programs had bugs in it. Tony Hoare was a great pioneer for proving the correctness of programs. But if you looked at the details ... I discovered from reading some of the articles, you know, I could find three bugs in a program that was proved

[3] "I looked up the record when I returned home and found that my memory was gravely flawed. Hoare had heard rumors about my work and he wrote to Stanford suggesting that I keep publication in mind. I replied to his letter on 16 November 1977—much earlier than I remembered." –D. Knuth

correct. [laughter] These were *small* programs. Now, he says, take my *large* program and reveal it to the world, with all its compromises. Of course, I developed TEX so that it would try to continue a history of hundreds of years of different ideas. There had to be compromises. So I was frightened with the idea that I would actually be expected to show someone my program. But then I also realized how much need there was for examples of good-sized programs, that could be considered as reasonable models, not just small programs.

I had learned from a Belgian man (I had met him a few years earlier, someone from Liège), and he had a system—it's explained in my paper on literate programming.[4] He sent me a report, which was 150 pages long, about his system—it was inspired by "The spirit in the machine". His 150-page report was very philosophical for the first 99 pages, and on page 100 he started with an example. That example was the key to me for this idea of thinking of a program as hypertext, as we would now say it. He proposed a way of taking a complicated program and breaking it into small parts. Then, to understand the complicated whole, what you needed is just to understand the small parts, and to understand the relationship between those parts and their neighbors.

In February of 1979, I developed a system called `DOC` and `UNDOC` ... something like the `WEB` system that came later. `DOC` was like `WEAVE` and `UNDOC` was like `TANGLE`, essentially. I played with `DOC` and `UNDOC` and did a mock-up with a small part of TEX. I didn't use `DOC` for my own implementation but I took the inner part called *getchar*, which is a fairly complicated part of TEX's input routine, and I converted it to `DOC`. This gave me a little 20-page program that would show the *getchar* part of TEX written in `DOC`. And I showed that to Tony Hoare and to several other people, especially Luis Trabb Pardo, and got some feedback from them on the ideas and the format.

Then we had a student at Stanford whose name was Zabala—actually he's from Spain and he has two names—but we call him Iñaki; Ignacio is his name. He took the entire TEX that I'd written in a language called *SAIL* (Stanford Artificial Intelligence Language), and he converted it to Pascal in this `DOC` format. TEX-in-Pascal was distributed around the world by 1981, I think. Then in 1982 or 1981, when I was writing TEX82,

[4] Pierre Arnoul de Marneffe, *Holon Programming*. Univ. de Liège, Service d'Informatique (December, 1973).

I was able to use his experience and all the feedback he had from users, and I made the system that became WEB. There was a period of two weeks when we were trying different names for DOC and UNDOC, and the winners were TANGLE and WEAVE. At that time, we had about 25 people in our group that would meet every Friday. And we would play around with a whole bunch of ideas and this was the reason for most of the success of TEX and METAFONT.

Another program I wrote at this time was called *Blaise*, because it was a preprocessor to Pascal. [laughter]

Dr. Petr Olšák: I have two questions.

What is your opinion of LATEX, as an extension of TEX at the macro level? I think that TEX was made for the plain TEX philosophy, which means that the user has read the *The TEXbook* ... [laughter] while LATEX is done with macros, and takes plain TEX as its base. And the second question: Why is TEX not widely implemented and used in commercial places. They use only mouse and WYSIWYG-oriented programs.

DEK: The first question was, what do I think about LATEX?

I always wanted to have many different macro packages oriented to different classes of users, and LATEX is certainly the finest example of these macro packages. There were many others in the early days. But Leslie Lamport had the greatest vision as to how to do this. There's also $\mathcal{AMS}$-TEX, and the mathematicians used Max Diaz's macros—I think it might have been called MaxTEX or something—in the early days before we had LATEX. Mike Spivak and Leslie Lamport provided very important feedback to me on how I could improve TEX to support such packages. I didn't want to ... I like the idea of a macro system that can adapt to special applications. I myself don't use LATEX because I don't have time to read the manual. [laughter] LATEX has more features than I need myself, in the way I do things. Also, of course, I understand TEX well enough that it's easier for me not to use high-level constructions beyond my control.

But for many people it's a simpler system, and it automates many of the things that people feel naturally ought to be automated. For me, the things that it automates are largely things that I consider are a small percentage of my total work. It doesn't bother me that I hand tune my bibliography, but it bothers other people a lot. I can understand why a lot of people prefer their way of working.

Also, when you're writing in a system like LaTeX you can more easily follow a discipline that makes it possible for other programs to find the structure of your document. If you work in plain TeX, you can be completely unstructured in your approach and you can defeat any possible process that would try to automatically extract bibliographic entries and such things from your document. If you restrict yourself to some kind of a basic structure, then other processes become possible. So that's quite valuable. It allows translation into other structures, languages and so on.

But I use TeX for so many different purposes where it would be much harder to provide canned routines. LaTeX is at a higher level; it's not easy to bend it to brand-new applications. Very often I find that, for the kind of things that I want to do, I wake up in the morning and I think of a project ... or my wife comes to me and says, "Don, can you make the following for me?" So I create ten lines of TeX macros and all of a sudden I have a new language specifically for that kind of a document. A lot of my electronic documents don't look like they have any markup whatsoever.

Now, your second question, why isn't TeX used more in commercial publication? In fact, I was quite pleasantly surprised to see how many commercial publishers in the Czech Republic are using TeX. Thursday night, I saw three or four Czech-English dictionaries that were done with TeX, and you know it's being used for the new Czech encyclopedia. And Petr Sojka showed me an avant garde novel that had been typeset with TeX with some nice tricks of its own very innovative page layout. In America, it's used heavily in legal publications, and behind the scenes in lots of large projects.

I never intended to have a system that would be universal and used by everybody. I always wanted to write a system that would be used for just the finest books. [laughter] Just the ones where the people had a more difficult than ordinary task, or they wanted to go the extra mile to have excellent typography. I never expected that it would compete with systems that are for the masses.

I'm not a competitive person, in fact. It made me very happy to think that I was making a system that would be primarily for mathematics. As far as I knew, there wasn't anybody in the world who would feel offended if I made it easier to typeset mathematics. Printers considered this to be "penalty copy", and it was something that they did only

grudgingly. They charged a penalty for doing this extra horrible work, to do mathematics. I never expected that I would be replacing systems that are used in a newspaper office or anything like that. It turned out that after I got going, we found we could make improvements ... in one experiment we re-typeset two pages of *Time* magazine, to show how much better it would be if they had a good line-breaking algorithm. But I never expected when I began that such magazines would ever use what I was doing because, well, it was a billion-dollar industry and I didn't want to put anyone out of work or anything.

So it was very disturbing to me in the early 80s when I found there was one man who was very unhappy that I invented TEX, because he had worked hard to develop a mathematical typesetting system that he was selling to people, and he was losing customers. So he wrote to the National Science Foundation in America, saying, "I'm a taxpayer and you're using my tax money to put me out of business." This made me very unhappy. I thought everything I was doing was for everybody's good. And here was a person I'd obviously hurt. But I also thought that I still should make TEX available to everyone, even though it had been developed with some help from the government. I don't think the government should give money only to things that are purely academic and not useful.

Yes?

?: I have a question about the usage of your typographic programs in commercial institutions like DTP studios and so on. I'd like to ask about using parts of the TEX source. You made clear that the programmers were free to incorporate parts of the TEX source into their own programs. There are some remarkable examples of this, do you know.

DEK: That question came up also last summer when I had a question and answer session at the TUG meeting in Florida.[5] I thought it would be fairly common to have special versions of TEX. I designed TEX so that it has many hooks inside; you can write extensions and then have a much more powerful TEX system readily adapted.

I guess I was thinking that every publishing house using TEX would have an in-house programmer who would develop a special version of TEX if they wanted to do an edition of the Bible, if they wanted to do an Arabic-to-Chinese dictionary or something. If they were doing an

[5] *TUGboat* **17**(1) (1996), pages 7–22.

encyclopedia, they could have their own version of TEX that would be used for this application.

A macro language is Turing-complete—it can do anything—but it's certainly silly to try to do everything in a high-level language when it's so easy to do it at the lower level. Therefore I built in hooks to TEX and I implemented parts of TEX as demonstrations of these hooks, so that a person who read the code could see how to extend TEX to other things. We anticipated certain kinds of things for chemistry or for making changebars that would be done in the machine language for special applications.

Certainly, if I were a publishing house, if I were in the publishing business myself, I would have probably had ten different versions of TEX by now for ten different complicated projects that had come in. They would all look almost the same as TEX, but no one else would have this program—they wouldn't need it, they're not doing exactly the book that my publishing house was doing.

That was what I thought would occur. And certainly, there was a point in the middle 80s when there were more than a thousand people in the world that knew the TEX program, that knew the intricacies of the TEX program quite well. They had read it, and they would have been able to make any of these extensions if they wanted. Now I would say that the number of people with a working knowledge of TEX's innards is probably less than a thousand, more than a hundred. It hasn't developed to the extent that I expected.

One of the most extensive such revisions is what I saw earlier this week in Brno—a student whose name is Thanh,[6] I think, who has a system almost done that outputs PDF format instead of `dvi` format. If you specify a certain flag saying `\PDFon`, then the output actually comes out as a file that an Acrobat reader can immediately read. I also expected that people would go directly to PostScript; that hasn't happened yet as far as I know.

No one has done a special edition of the Bible using TEX in the way I expected. There were some extensions in Iceland; I don't remember if they did it at the higher level—I think they did it mostly at the macro level, or maybe entirely.

[6] Han The Thanh; see Petr Sojka, Han The Thanh and Jiří Zlatuška, "The Joy of TEX2PDF—Acrobatics with an alternative to DVI format", *TUGboat* **17**(2) (1996).

Anyway, I made it possible to do very complicated things. When you have a special application, I was always expecting that you would want to have a specially tuned program there because that's where it's easiest to do these powerful things.

?: I want to ask which features of TEX were in the first version—for example, line-breaking, hyphenation, and macro processing—if all these things were in the first version?

DEK: The very first version was designed in April 1977. I did have macros and the algorithm for line-breaking. It wasn't as well developed; I didn't have all the bells and whistles like `\parshape` at that time, but from the very beginning, from 1977 on, I knew I would treat the paragraph as a whole, not just line by line. The hyphenation algorithm I had in those days was not the one that we use now; it was based on removing prefixes and suffixes—it was a very peculiar method, but it seemed to catch about 80% of the hyphens. I worked on that just by looking at the dictionary: I would say, the word starts with "anti", then put a hyphen after the "i"; and similarly at the end of the word. Or if you have a certain combination of letters in between, in the middle, there were natural breaks. I liked this better than the *troff* method, which had been published. The hyphenation algorithm is described in the old TEX manual, which you can find in libraries.[7]

Now, you said the line-breaking, hyphenation, macros, ... I developed the macro language in the following way. I took a look at Volume 2 of *The Art of Computer Programming* and I chose representative parts of it. I made a mock-up of about five pages of that book, and said, "How would I like that to look in a computer file?" And that was the whole source of the design.

I stayed up late one night and created TEX. I went through Volume 2 and fantasized about natural-looking instructions—here I'll say "backslash algorithm", and then I'll say "algorithm i", and then I'll say "algstep", you know. This gave me a little file that represented the way I wanted the input to look for *The Art of Computer Programming*. The file also included some mathematical formulas. It was based on the idea of *eqn*; the *troff* language had shown me a way to represent mathematics that secretaries could learn easily. And that was the design. Then I had to implement a macro language to support those features.

[7] *TEX and METAFONT: New Directions in Typesetting* (cited in footnote 1).

The macro language developed during 1978, primarily with the influence of Terry Winograd. Terry was writing a book on linguistics, a book on English grammar. He wanted to push macros much harder than I did, and so I added `\xdef` and fancier parameters for him.

The hyphenation algorithm we have now was Frank Liang's Ph.D. research. He worked with me on the original hyphenation method, and his experience led him to discover a much better way, which can adapt to all languages—I mean, to all western languages, which are the ones that use hyphens.

As far as the spacing in mathematics is concerned, I chose three standards of excellence of mathematical typesetting. One was Addison-Wesley books, in particular *The Art of Computer Programming*. The people at Addison-Wesley, especially Hans Wolf, their main compositor, had developed a style that I had always liked best in my textbooks in college. Secondly, I took *Acta Mathematica*, from 1910 approximately; this was a journal in Sweden ... Mittag-Leffler was the editor, and his wife was very rich, and they had the highest budget for making quality mathematics printing. So the typography was especially good in *Acta Mathematica*. And the third source was a copy of *Indagationes*, the Dutch journal. There's a long fine tradition of quality printing in the Netherlands, and I selected an issue from 1950 or thereabouts, where again I thought that the mathematics was particularly well done.

I took these three sources of excellence and I looked at all the mathematics formulas closely. I measured them, using the TV cameras at Stanford, to find out how far they dropped the subscripts and raised the superscripts, what styles of type they used, how they balanced fractions, and everything. I made detailed measurements, and I asked myself, "What is the smallest number of rules that I need to do what they were doing?" I learned that I could boil it down into a recursive construction that uses only seven types of objects in the formulas.

I'm glad to say that three years ago, *Acta Mathematica* adopted TEX. And so the circle has closed. Addison-Wesley has certainly adopted TEX, and I'm not sure about the Dutch yet—I'm going to visit them next week. [laughter] But anyway, I hope to continue the good old traditions of quality.

I have to call on people who haven't spoken. George...

Jiří Veselý: I have a question. You are asked every time carefully

regarding all suggestions and things like that for improvements. Once I was asked about the possibility to make a list of all hyphenated words in the book. I was not able to find in your book a way to do this. I would like to know something about your philosophy what to include and what not to include. What would be in that special package, and what would be in TEX?

DEK: The question is, what is the philosophy that I use to try to say what should be a basic part of TEX and what should be harder to do or special, or something like that. Of course, these decisions are all arbitrary. I think it was important, though, that the decisions were all made by one person, even though I'm not ... I certainly make a lot of mistakes. I tried the best to get input from many sources, but finally I take central responsibility to keep some unity. Whenever you have a committee of people designing a system, everyone in the committee has to feel proud that they have contributed something to the final language. But then you have a much less unified result because it reflects certain things that were there to please each person. I wanted to please as many people as I could but keep unity. So for many years we had a weekly meeting for about two hours every Friday noon, and we had visitors from all over the world who would drop in. We would hear their comments and then we would try to incorporate the ideas that we heard during that time.

Now you ask specifically about why don't we have an easy way to list all the hyphenations that were made in the document. It sounds like a very nice suggestion, which I don't recall anyone raising during those weekly meetings. The words that actually get hyphenated, the decision to do that is made during the *hpack* routine, which is part of the line-breaking algorithm. But the fact that a hyphenation is performed by *hpack* doesn't mean that it's going to appear in the final document, because you could discard the box in which this hyphenation was done.

It's very easy in TEX to typeset something several times and then choose only one of those for the actual output. So, to get a definitive representative of the hyphenation, you'd have to catch it in the output routine, where the discretionary had appeared. This would be easy to do now in a module specially written for TEX. I would say that right now, in fact, you could get almost exactly what you want by writing a filter that says to TEX "Turn on all of the tracing options that cause it to

print, to give the page contents." Then a little filter program would take the trace information through a UNIX pipe and it would give you the hyphenated words. It would take an afternoon to write this program, maybe two afternoons ... and a morning. [laughter] You could get that now, but it was not something that I can recall I ever debated whether or not I should do at the time we were having these weekly discussions on TEX.

My paper on "The errors of TEX" has the complete record of all the changes that were made since 1979, with dates, and with references to the code, exactly where each change appears. And so you can see the way the evolution was taking place. Often the changes would occur as I was writing *The TEXbook* and realizing that some things were very hard for me to explain. I would change the language so it would be easier to explain how to use it. This was when we were having our most extensive meeting with users and other people in the group as sources of ideas; the part of the language I was writing about was the part that was changing at the moment.

During 1978, I myself was typesetting Volume 2, and this led naturally to improvements as I was doing the keyboarding. In fact, improvements occurred almost at a steady rate for about 500 pages: Every four pages I would get another idea how to make TEX a little better. But the number of ways to improve any complicated system is endless, and it's axiomatic that you never have a system that cannot be improved. So finally, I knew that the best thing I could do would be to make no more improvements—this would be better than a system that was improving all the time.

In fact, let me explain. As I was first developing TEX at the Stanford Artificial Intelligence Laboratory, we had an operating system called WAITS, which I think is the best that the world has ever seen. Four system programmers were working full time making improvements to this operating system. And every day that operating system was getting better and better. And every day it was breaking down and unusable.

In fact I wrote the first draft of *The TEXbook* entirely during downtime. I would take my tablet of paper to the Artificial Intelligence Laboratory in the morning and I would compute as long as I could. Then the machine would crash, and I would write another chapter. Then the machine would come up and I could type a little bit and get a little more done. Then, another hang up; time to write another chapter. Our

operating system was always getting better, but I couldn't get much computing done.

Then the money ran out; three of the programmers went to Lawrence Livermore Laboratory and worked on a new operating system there. We had only one man left to maintain the system, not to make any more improvements. And it was wonderful! [laughter] That year, I could be about as productive as anyone in the world.

So I knew that eventually I would have to get to the point where TEX would not anymore improve. It would be steady and reliable, and people would understand the warts it had ... the things that it doesn't do.

I still believe it's best to have a system that is not a moving target. After a certain point, we need something that is stable, not changing at all. Of course, if there's some catastrophic scenario that we don't want ever to happen, I still change TEX to avoid potential disasters. But I don't put in nice ideas any more.

Of course, there are other people working on extensions to TEX that will be useful for another generation. And they will also be well advised at a certain point to say "Now we will stop, and not change our system any more." Then there will be a chance for another group later.

Dr. Karel Horák: I'd like to ask about the idea of the italic font in mathematics. I never saw other textbooks that use different fonts for italics in text and in mathematics, so I'm asking if it's your own idea or if it comes also from these three sources?

DEK: That's right. I didn't find in any of the other books the idea of having a text italic and a math italic. I wanted the math italic to look as beautiful as possible, and I started with that. But then I found that the text italic was not as good, so I had METAFONT and it was easy to get text italic that would look better. If I made the text italic good, then the math would not position the subscripts and the superscripts as well.

It's partly because of what I explained before—TEX has only four numbers to go with every character. Printers, in fact, in the old days, had only three numbers; they didn't have the italic correction. So they couldn't do even that much automatically; the better printers adjusted mathematical spacing by hand. But italic now, the italic fonts of today by all the font designers are much better than they used to be. We've

seen a great improvement in italic typography during the last ten, fifteen years. In fact, if you read older books you'll sometimes say, "How could anybody read this italic?", or "Why did they accept such peculiar spacing?", but it was based on the constraints of metal type. The whole idea of italic correction was not in any other book, but it was necessary for me to get the spacing that I wanted.

When I showed type designers mathematical formulas, they could never understand why mathematicians want italic type in their formulas. It seems you're combining a roman 2 with an italic x. And they said, "Wouldn't the positioning be so much simpler if you had a regular, non-sloped font in mathematics?" And I think it was Jan van Krimpen, who worked with a Nobel Prize physicist in the Netherlands, in Haarlem—what was his name?[8] I think he was the second person to receive the Nobel Prize in physics; he died in the 20s—anyway he and van Krimpen were going to develop a new font for mathematics in the Netherlands, and it wasn't going to have italics for mathematics. It was going to be unified between the Greek letters and other symbols that mathematicians wanted. But the project stopped because the physicist died; van Krimpen finished only the Greek, which became fairly well used.

Several other font designers have visited Stanford. When they looked at mathematics, they said, "Well, why don't you use a non-sloping font?" Hermann Zapf made a proposal to the American Mathematical Society that we would create a new typeface for mathematics which would include the Fraktur alphabet, and Greek, and script, and special characters, as well as ordinary letters. One key idea was that it would not have sloped characters, so that x would be somehow straight up and down. Then it should be easier to do the positioning, the balancing. Hermann created a series of designs, and we had a large committee of mathematicians studying the designs and commenting on them and tuning them.

This font, however, proved to be too radical a change for mathematicians. I've seen mathematicians actually writing their documents where they will write an x slanted twice as much—I mean, they make it look very italic; it looks like a mathematical letter to them. So after 300 years of seeing italic math in print, it's something that we feel is right.

[8] It was H.A. Lorentz. See John Dreyfus, *The Work of Jan van Krimpen* (London: Sylvan Press, Museum House, 1952), page 28.

There are maybe two dozen books printed, well, maybe more, maybe a hundred, printed with the AMS Euler font, but most mathematicians think it's too different.

I find now that the Euler Fraktur font is used by almost everyone. In Brno, I saw Euler Roman used as a text font for a beautiful book, a translation of Durer's *Apocalyse* in Czech. I also saw it a few days ago in some class notes. Once, when I was in Norway, I noticed that everyone's workstation was labeled with the workstation's name in AMS Euler, because people liked it. It's a beautiful font, but it hasn't been used as the typeface for mathematics in a large number of books.

Dr. Karel Horák: If there are no other questions, I would thank Professor Knuth very much for this session. [wide prolonged applause]

DEK: Thank you all for excellent questions.

Dr. Karel Horák: [Closing comments in Czech.]

Vizitky v LaTeXu

Josef Barák

Vizitky, čili navštívenky, se staly v posledních letech běžným doplňkem. Jejich návrhem a výrobou se zabývá mnoho firem. Vizitka navržená firmou je však jednak poměrně drahá (cca 1,50 Kč a více za kus), jednak si nelze objednat menší sérii než padesát kusů (a pokud ano, je účtován příplatek). Navíc se představa návrháře nemusí vždy zcela shodovat s představou zákazníka.

Po zvážení všech výše uvedených důvodů jsem se rozhodl, že budu vizitky sobě i svým známým vyrábět sám. Protože i vizitka by měla splňovat určitá estetická kritéria a protože se již nějakou dobu zabývám TeXem, přesněji řečeno LaTeXem, byla volba vhodného programu jednoznačná.

Vzhledem k tomu, že mám přístup k Internetu, nejdříve ze všeho jsem se pídil po vhodném makru, které by výrobu vizitek nějak ulehčilo a zautomatizovalo. Moje snaha byla částečně úspěšná, neboť jsem nalezl balík

BUSCARD.ZIP[1] od Rudy Blažka. V souboru CARDS.TEX obsahuje makro, pomocí něhož lze poměrně snadno vytvořit deset vizitek (dva sloupce po pěti) na jeden list papíru, a to včetně ořezových značek. Pro lepší návrh je zde také možnost nechat si zobrazit vizitky včetně rámečků okolo nich, tj. v prohlížeči lépe vidíme, jak bude hotová vizitka vypadat. Tisk vizitek i s rámečky by však nebyl vhodný, proto lze zobrazení rámečků vypnout.

Vyzkoušel jsem demonstrační soubor a poté se pustil do vytváření vlastních vizitek a také vizitek pro své známé. Zde jsem však narazil na několik omezení:

- je počítáno se standardní velikostí papíru $8{,}5 \times 11$ palců
- velikost jedné vizitky je dosti nestandardní ($8{,}7 \times 5{,}1$ cm)
- nelze pohodlně vložit obrázek (logo apod.)
- v demonstračním souboru ani nikde jinde není zmínka o způsobu vkládání čarové grafiky (rámečky, linky apod.)
- nelze kromě adresy zaměstnání a soukromé vytisknout ještě adresu třetí, což je v některých případech také potřebné (lékař, který má ordinace na více místech apod.)

Rozhodl jsem se tedy makro upravit podle svých potřeb. Odstranil jsem všechny výše uvedené nedostatky (i když možná ne vždy nejefektnějším způsobem), a protože se domnívám, že by takto upravené makro využilo více lidí, rozhodl jsem se dát jej k dispozici veřejnosti.

Upravené makro se (i s novým demonstračním souborem a krátkým popisem) nachází v balíku VIZITKY.ZIP[2]. Vlastní makro je ve dvou variantách – v souboru A4CARDS.TEX pro klasickou vizitku se dvěma adresami a v souboru A4CARDS3.TEX pro vizitku se třemi adresami (více se jich na vizitku daného rozměru nevejde).

Rozdíly oproti makru původnímu:

Makro v souboru A4CARDS.TEX:

- standardní velikost papíru je nyní A4
- jako velikost vizitky byla zvolena nejběžnější 9×5 cm

[1] ftp://ftp.muni.cz/pub/tex/local/styles/packages/buscard.zip
[2] ftp://ftp.cstug.cz/pub/tex/local/cstug/barak/vizitky/vizitky.zip

- byla doplněna makra `\vlozobr` a `\jenobr` pro práci s obrázky ve formátu `PCX`
- vše bylo zdokumentováno v novém demo souboru, a to včetně použití nových maker, možností vkládání čarové grafiky apod.

Makro v souboru `A4CARDS3.TEX`:

- shodné s předchozím a navíc s možností tisku tří adres (nebo jakýchkoliv informací ve třech sloupcích)

Nevýhody upravených maker:

- jsou podporovány obrázky ve formátu `PCX` (s využitím příkazu `\special`), z čehož vyplývá určitá závislost jak na konkrétní implementaci TEXu, tak i na typu počítače – vše bylo odzkoušeno pro CSTEX na PC s MS-DOSem; nehodláme-li možnosti vkládání obrázků využít, měla by být funkčnost makra neomezená
- je podporován především LATEX (definice je pomocí `\newcommand` apod.), což vychází již z původního makra; patřičná úprava do `plain`TEXu by však neměla být problémem

Popis souborů balíku `VIZITKY.ZIP`

- A4CARDS.TEX — upravené a rozšířené makro
- A4CARDS3.TEX — stejné jako výše uvedené, ale navíc s možností tisku adres ve třech sloupcích
- VIZITKY.TEX — demonstrační soubor pro A4CARDS.TEX
- VIZITKY3.TEX — dtto pro A4CARDS3.TEX
- VIZITKY.DOC — krátký popis ve formátu ASCII v češtině v kódování Kamenických
- LOGO.PCX — obrázek využívaný v demonstračních souborech

Tisk

je v tomto případě největším kamenem úrazu. Na běžné laserové tiskárně lze totiž tisknout jen na běžný (tj. měkký) papír, což pro vizitky není zrovna nejvhodnější. Tento problém lze ovšem řešit dvěma způsoby:

1. tisknout na běžný papír a pak si vše nechat zkopírovat na papír tvrdý (v takovém případě pak stojí deset vizitek něco kolem 6 Kč) nebo lépe

2. někde sehnat laserovou tiskárnu, která přímo umožňuje tisk na vhodný papír, provést tisk na disketu do souboru `.HP`, popř. `.PS` a nechat si tento soubor vytisknout

První způsob je poněkud méně vhodný proto, že při kopii může dojít k částečnému zhoršení kvality výsledku. Dle mých zkušeností však i vizitka vytisknutá v rozlišení 300 dpi a následně zkopírovaná vypadá velmi přijatelně (nesmí se zde ovšem vyskytovat extrémně malé písmo, jemná grafika apod.). Ke druhému způsobu bych měl ještě následující poznámku: přineseme-li disketu se souborem `.HP` nebo `.PS` na počítač s vhodnou tiskárnou, na němž ale není nainstalován TEX, nejjednoduššeji provedeme tisk příkazem MS-DOSu: `copy` *soubor* `PRN /B`

Důležité je uvést parametr `/B`, což značí binární kopii. V tom případě se do tiskárny pošle vše, bez ohledu na případné znaky `EOF` a jim podobné.

Přeji všem tvůrcům vizitek v TEXu mnoho úspěchů.

Josef Barák
xbarak00@kpve.fee.vutbr.cz

Recenze knihy „Typografický systém TEX“

Antonín Strejc

Název: Typografický systém TEX
Autor: RNDr. Petr Olšák
Počet stran: 270
Vydalo: Československé sdružení uživatelů TEXu
Vydání: první
Cena: 80 Kč pro členy sdružení, 150 Kč pro ostatní

Když před časem Petr Olšák ohlásil dokončení své knihy o TEXu, přijal jsem tuto zprávu s velkou radostí. Konečně budeme mít (rozuměj my Češi a Slováci) stravitelnější studijní materiál o TEXu, než pro

mnohé přece jen nepříliš dietní páně Knuthův TEXbook. Docela rád jsem v duchu přeháněl a pohrával si s idealistickou představou, jak se wokenní a myší otroci začnou převracet v TEXperty, sem tam mezi nimi uzraje i nějaký ten wizardíček. Ze začátečníků, teprve těžce a klopotně se probojovávajících do společenství TEXistů se snadno a rychle stanou brilantní komponisté těch nejšílenějších maker a artisté balancující bravurně na nitkách všech těch dérty triků a jiných nepochopitelností. Díky tomuto všemu pak těžce nemocná stará dáma jménem typografie opět povstane z popela jako bájný pták Fénix.

Pokud jste snad jen na okamžik uvěřili, že něco takového je možné, třebaže ne v tak absurdní míře, kterou jsem právě popsal, raději si knížku vůbec nekupujte, budete zklamáni. Přesto, zůstanete-li na zemi a řeknete-li si, že se chcete dozvědět něco o TEXu jako takovém, i o spoustě věcí, které s ním tak či onak souvisejí, knížku si určitě rychle obstarejte. Určitě nebudete litovat, tak jako nelituji já, i když k některým věcem mám výhrady. Pokusím se vás nyní se svými názory blíže seznámit.

O čem kniha není a o čem tedy vlastně je

Přiznám se, že jsem se těšil na to, že knížka bude více o tom, jak pracovat v TEXu. Ona je ale spíše o tom, jak pracovat s TEXem. Přeloženo do srozumitelnější řeči to znamená, že se autor více zabývá systémem z pohledu uživatele, který nesmí ztrácet souvislosti, než očima programátora, ale to byl asi záměr. Kniha by se rozhodně nemohla jmenovat Programování v TEXu (takovou bych velice rád vlastnil, ale napsat opravdovou a dobrou učebnici programování v čemkoli je moc a moc těžká věc). Charakter díla je víc popisný než didaktický, mapuje soudobý TEXovský prostor a zavede vás v něm určitě i do nějakého kouta, který uznáte za vhodné poznat blíž. K tomu napomohou i četné odkazy na patřičnou literaturu zabývající se pak zvolenou tématikou podrobněji. Záběr knihy je tedy spíše širší než hlubší a při vážném zájmu čtenáře o TEX představuje podle mne dobrý odrazový můstek k dalšímu bádání.

Jednotlivé části knihy

První dvě kapitoly knihy jsou, jak autor píše v předmluvě, určené naprostým začátečníkům, kteří nevědí o TEXu nic. Tyto kapitoly představují podle mého názoru nejslabší část díla. Hned na samém začátku autor předloží čtenáři příklad TEXovského zdrojového textu (dopis), ukáže, jak to vypadá, když je tento text zpracován TEXem. Až potud by se dalo říci dejme tomu. Dále pak dopis postupně vylepšuje zařazováním formátovacích příkazů, simuluje syntaktickou chybu, odstraňuje ji, až dojde k jakési konečné formě. Toto je již na začátečníka trochu silná káva. Příkazům nerozumí a marně zírá na složené závorky, backslashe, chybová hlášení a další věci. V kapitole se pracuje s pojmy jako tiskové zrcadlo, plainTEX, TEXovské makro atd. bez jakéhokoli předchozího vysvětlení. Vysvobozením pro zmateného laika je až závěrečný odkaz na jinou literaturu pro zvládnutí prvních krůčků s TEXem. Teprve po této, až neuvěřitelně konkrétní kapitole následuje kapitola obecně pojednávající o tom, co to vlastně TEX je a jaké má vlastnosti. I zde se však zničehonic odbíhá ke konkrétnostem bez vysvětlení. Například detailně popisovaný mechanismus tvorby poznámek pod čarou nebo ukázka sazby šachového diagramu s vysvětlením, že není nutné sázet jej ručně, ale stačí napsat příkaz `\diag`. Skoro to až svádí k tomu zkusit si v TEXu ten příkaz napsat… Dal jsem tyto dvě kapitoly k přečtení několika lidem, kteří o TEXu do té doby nevěděli téměř nic, a výsledkem bylo jen jejich zmatení, a nikoli průprava pro četbu dalších kapitol. Pro příští vydání bych proto doporučoval úplné vypuštění zejména první kapitoly a nahrazení nějakou větičkou v úvodu, že se předpokládá jistá minimální zkušenost s TEXem. Ponechal bych jen odkazy na jiné publikace pro zvládnutí prvních krůčků.

Další kapitoly však již stojí za přečtení. Kapitola 3 popisuje nejpoužívanější TEXovské formáty, v další kapitole je bližší zmínka o implementacích TEXu pod operačními systémy UNIX a MS-DOS. Pátá kapitola pojednává o problému češtiny v TEXu, šestá o fontech. Sedmá kapitola, která však má k mému velkému zklamání jen třináct stran, lehce naznačí možnosti programovacího jazyka TEXu. Následuje poměrně rozsáhlá a velmi zajímavá osmá kapitola o METAFONTu. Důležitá je i devátá kapitola o návaznosti TEXu na jazyk PostScript. Kapitola 10 velmi přehledně shrnuje možnosti zařazování obrázků do TEXovských dokumentů. Dále následuje kapitola o nástrojích pro tvorbu rejstříku a citací. V ka-

pitole 12 zaujme zejména zmínka o WEBu, nástroji, který byl použit pro napsání zdrojových textů TEXu a METAFONTu.

Kniha obsahuje ještě dodatky (značené písmeny A až G) zahrnující slovníčky TEXu a METAFONTu, seznam a stručný popis souborů a balíků týkajících se TEXu a vystavovaných v Internetu, dále seznam literatury, konkrétní ukázky formátování, fontové tabulky a rejstřík.

Chyby a nepřesnosti v knize

Kniha obsahuje velké množství informací na omezeném rozsahu 270 stran, a asi právě vzhledem k této velké hustotě informace se vydání neobešlo bez některých nepřesností (překlepy v názvech příkazů, gramatika, stylistika atd.), přestože text údajně prošel několika korekturami. Procento těchto chyb však naštěstí není velké, autor navíc průběžně zjišťované chyby zaznamenává a zveřejňuje je formou opravenky, která je dostupná na `ftp.math.feld.cvut.cz` v adresáři `/pub/olsak/tst`, soubor `BUGS`. Tento soubor je také dostupný přes `http://math.feld.cvut.cz/olsak` v položce `TST` a `Errata`.

Zpracování knihy

Pokud jde o typografické zpracování, je velmi precizní, i když je z úpravy knihy i jejího členění cítit inspirace Knuthovým TEXbookem. To se sice někomu nemusí zdát příliš originální, nicméně se zde autorovi nedá z hlediska obecných typografických norem snad vůbec nic vytknout.

Patrně velmi kvalitně byl proveden osvit, neboť použité písmo Computer Modern je výborně čitelné a ukazuje se, že přes časté výhrady tiskařů či pracovníků DTP studií k těmto fontům se dá práce odvést v dobré kvalitě (často bývá CM fontům vytýkána přílišná vlasovost).

Knihařské zpracování publikace je však už poněkud horší. Neobvyklý je formát knížky (155×225 mm), nevím, zda je to úmysl nebo důsledek špatného ořezu knihy. Absolutně chybný je výběr vazby – tzv. lepený hřbet. Budete-li brát knížku do ruky každý den, brzy se rozpadne a vy budete mít starost, co s jednotlivými listy (v takovém případě doporučuji kroužkovou či spirálovou vazbu torza knihy). Zde se podle mého názoru šetřilo na naprosto nevhodném místě a mnohé ohlasy čtenářů brzy po zakoupení knihy to jen potvrzují.

Závěrem

Knihu Typografický systém TEX nepochybně stojí za to mít ve své knihovně. Přes některé drobné nedostatky (na jejichž závažnost však mohou mít čtenáři jiný názor než já) představuje dobrý a kompaktní soubor užitečných informací, zhuštěných do relativně malého počtu stran, spolu s odkazy na podrobnější informace, které čtenář nalezne v četných dalších publikacích.

Pro případné další vydání doporučuji autorovi zvážit, zda úvodní pasáže zapadají do celkového rámce knihy. Pokud snad opravdu bylo záměrem přisoudit knize i úlohu zprostředkovatele nejprvnějších informací o TEXu nutných pro četbu dalších kapitol, je možné o úspěchu tohoto záměru pochybovat. Četl jsem kdesi autorovo vyjádření, že knihu možná budou kupovat laici jen kvůli prvním kapitolám a pokročilí kvůli zbytku. Potom se mi ale zdá zbytečné, aby si obě skupiny čtenářů musely kupovat i tu část, která je nezajímá, a raději bych látku rozdělil do dvou samostatných dílů. Tím by vzniklo i více prostoru pro vysvětlování, které by nemuselo být tolik zhuštěné.

Rozhodně bych se přimlouval, ať už pro případ druhého vydání této knihy či jiných autorem plánovaných publikací, pro upuštění od lepené vazby a jejího nahrazení odolnější vazbou šitou.

AZBTOTEX — program pro převod azbuky z T602 do TEXu

Josef Barák

Úvod

K vytvoření programu AZBTOTEX[1] mě inspiroval dotaz, zda lze v TEXu psát azbukou. Odpověděl jsem, že samozřejmě ano. Protože jsem však nevěděl přesně, jak na to, daný problém jsem prostudoval. Zjistil jsem,

[1] `ftp://ftp.cstug.cz/pub/tex/local/cstug/barak/azbtotex`

že s vhodnými fonty (používám CSTEX) by ani pro azbuku problémy nebyly. Problém však nastává s klávesnicí. Abychom vytiskli patřičný ruský znak, je nutné použít mezinárodní transliteraci azbuky. Je tedy nutné psát např. „zashchita“, „ya reshil“, apod. Tímto způsobem lze docela dobře napsat jeden řádek textu, hůře by se však vytvářel několikastránkový dopis. Zamyslel jsem se tedy nad tím, jak zdrojový text pořídit lépe. Vcelku příjemně definované hned dvě ruské klávesnice (jednu z nich s rozložením co nejpodobnějším *českému* psacímu stroji) má známý textový editor T602. Vytvořil jsem tedy preprocesor, který převede zdrojový text, pořízený v T602, do tvaru vhodného pro zpracování LATEXem a po nepatrných úpravách i `plain`TEXem.

Použitím programu AZBTOTEX spojujeme výhodu snadného pořízení ruského textu v T602 a vysoké typografické kultury systému TEX.

Program AZBTOTEX

Jak pořídit zdrojový text v T602

Textový editor T602 spustíme s parametrem **/r** (nebo můžeme použít dávkový soubor `T602R.BAT`).

K oběma ruským klávesnicím se dostaneme pomocí nabídky `Text602` a podnabídky `Klávesnice..` – zaškrtnutím č. 7 (`RUS`) přepneme na klávesnici s rozložením jako na ruském psacím stroji (pro většinu cizinců nepříliš vhodné). Zaškrtneme-li však klávesnici č. 8 (`KL8` – tedy klávesnici uživatelskou), máme (v případě, že jsme ji nepředefinovali) ruskou klávesnici s rozložením téměř jako na českém psacím stroji. Nápověda k rozložení libovolné klávesnice se v T602 zobrazí pomocí `Alt-X`.

Pořízený dokument uložíme běžným způsobem. Před uložením se však přesvědčíme, že máme zapnuto kódování Kamenických (v T602 lze i azbuku kódovat dvojím způsobem, podobně jako češtinu). Kódování se přepne pomocí nabídky `Text602\Vst/výs. kód..` zaškrtnutím volby č. 1 (`KEYBCS2`). Pomocí stejné nabídky lze překódovat již uložený dokument.

Jak převést pořízený text do TEXu

Nyní nastává okamžik, kdy bude vhodné použít preprocesor. Syntaxe je velmi jednoduchá:

`AZBTOTEX vstup [výstup]`

Vstupem může být soubor ve formátu T602, popřípadě i v ASCII (export z T602). Formát je rozpoznán automaticky.

V případě formátu T602 je automaticky rozpoznáno i kódování (pokud není Kamenických – viz výše – je doporučen další postup).

Převádíme-li přímo z ASCII, je situace složitější. V tomto případě totiž nelze (jednoduše) rozpoznat typ kódování. Jestliže tedy konverzí vznikne něco nesrozumitelného, bylo použito jiné kódování a je vhodné původní text překódovat pomocí T602. Rovněž nelze jednoznačně určit, kde končí odstavce. Je proto nutné ve výsledném textu odstavce vytvořit ručně. Poslední nepříjemností je ztráta informací o různých typech písma (tučné písmo, kurzíva apod.), protože při exportu z T602 do ASCII se všechny formátovací znaky ztrácejí.

Poznámky:

- Z těchto důvodů doporučuji používat výhradně přímo formát T602. Možnost konverze z ASCII slouží pro případ, že máme k dispozici soubor pouze v tomto formátu.
- Vstupní soubor může mít libovolnou příponu (nemusí to být pouze `TXT` či `602`) kromě přípony `TEX`. Není-li přípona zadána, doplní se `TXT`.

Výstupem je soubor ve formátu, který je přímo přeložitelný LaTeXem. Po drobných úpravách (vymazání `\begin{document}`, `\end{document}` atd.) je přeložitelný i `plain`TeXem, popř. dalšími formáty TeXu. Není-li výstup explicitně zadán, dosadí se název stejný jako u vstupu, ale s příponou `TEX`.

U vstupu i výstupu samozřejmě lze, kromě názvů těchto souborů, zadat i disk, popřípadě celou adresářovou cestu, je-li některý ze souborů v jiném adresáři než aktuálním.

Co dál s převedeným textem

Výsledný soubor s převedeným textem lze v TeXu zpracovat obvyklým způsobem. Vzhledem k tomu, že fonty s azbukou nejsou běžně používány, pravděpodobně se před prvním prohlížením spustí `MfJob`, který bude chtít vše potřebné vygenerovat. To mu samozřejmě povolíme.

Kromě transliterace azbuky jsou vhodně převedeny i odstavce a rovněž jsou odstraněny formátovací znaky T602. Také je převedena azbuka

tučná, kurzíva, široká, vysoká a velká. Vše je vyzkoušeno na běžné instalaci CSTEXu (v podobě šířené CSTUGem).

Protože předpokládám nejčastější zpracování v LATEXu, je výsledný soubor upraven tak, že lze přímo provést překlad. Jsou tedy automaticky doplněny takové příkazy jako `\documentstyle`[2], `\begin{document}` a `\end{document}`. Pro lepší typografický vzhled výsledného dokumentu je však vhodné provést ruční doformátování.

Hardware a software

Program AZBTOTEX je určen pro počítače PC-kompatibilní s operačním systémem MS-DOS. V případě potřeby lze spustit i z prostředí Windows.

Přeji všem uživatelům mého programu AZBTOTEX pouze příjemné zážitky při práci s tímto programem. V případě problémů, nejasností nebo s případnými náměty se můžete obracet na moji e-mailovou adresu.

Literatura:

[1] Olšák, P.: Typografický systém TEX, Československé sdružení uživatelů TEXu, Praha 1995.
[2] ČSN 01 0185 – Transliterace cyriliky.

Josef Barák
xbarak00@kpve.fee.vutbr.cz

[2]Lze samozřejmě použít i `\documentclass`

Problémy s fonty v TEXu

Arnošt Štědrý

Poslední dobou roste zájem o použití *jiných* fontů v TEXu. Tomuto problému bylo věnováno již několik článků ve Zpravodaji (např. [1, 3, 4, 6]). Všechny však vycházely z premisy, že zpracovávaný materiál (font) je po formální stránce bezchybný. Z vlastní dlouhé praxe vím, že tomu tak není a chyby ve fontech, znemožňující jejich použití, jsou velice záludné. Úkolem tohoto příspěvku by mělo být položení teoreticko-praktického základu k řešení některých potíží.

Tedy abych byl přesný, co myslím *jiným* fontem: jedná se o PostScriptové fonty Type1, Type3, případně TrueTypy. METAFONTové fonty považuji za natolik srostlé s TEXem, že ani do článku o potížích nepatří.

Zaměřme se nyní na PostScriptové fonty, TrueTypy odbudeme nakonec krátkou poznámkou o konverzích.

Co budeme potřebovat k TEXovské sazbě

Každý uživatel TEXu ví, nebo si četbou např. [2] doplní, že TEX má informaci o fontech rozdělenou do dvou komponent. První část – metrická – je uložena v TFM souborech, podle ní TEX počítá sazbu. Druhá obsahuje tvary písmenek a její formát je závislý na použitém ovladači (typicky jsou to pk-fonty).

TEXovské metriky

Metriky jsou soubory s koncovkou TFM (zkratka od TEX Font Metrics) – podívejme se trochu zblízka, co obsahují. Programem `tftopl` je převedeme do čtivé podoby, můžeme se v nich i trochu porýpat a pak programem `pltotf` převést zpět. Pamatujme však, že provedeme-li změny, jedná se již o zcela jiný font, což by se mělo projevit i v názvu.

Takže nejprve hlavička:

```
(FAMILY CMR)
(FACE O 352)
```

```
(CODINGSCHEME TEX CS TEXT)
(DESIGNSIZE R 10.0)
(COMMENT DESIGNSIZE IS IN POINTS)
(COMMENT OTHER SIZES ARE MULTIPLES OF DESIGNSIZE)
(CHECKSUM O 15736045506)
(FONTDIMEN
   (SLANT R 0.0)
   (SPACE R 0.333334)
   (STRETCH R 0.166667)
   (SHRINK R 0.111112)
   (XHEIGHT R 0.430555)
   (QUAD R 1.000003)
   (EXTRASPACE R 0.111112)
   )
```

Stručně řečeno hlavička obsahuje základní informace o fontu – do které rodiny patří, jak je kódován, kontrolní součet (pokud byl font vytvořen METAFONTem, je to informace pro ovladač, zdali byly pk-soubory stvořeny ze stejného zdrojového kódu), a nakonec parametry fontu, ke kterým lze přistupovat v TEXu pomocí `\fontdimen`.

SLANT Sklonění fontu: odtud se čerpá informace o umístění akcentů a kurzívních korekcích.
SPACE Normální mezera mezi slovy.
STRETCH Jak se může tato mezera roztáhnout.
SHRINK Nebo stáhnout.
XHEIGHT Výška písmene X (opět kvůlivá akcentům) v TEXu jednotka ex.
QUAD Tohle je emsize neboli čtverčík, v TEXu em, tedy šířka písmene M.
EXTRASPACE Mezera, která se dává mezi větami v anglofonních oblastech.

V matematických fontech je v této části údajů mnohem více.

Následuje informace o ligaturách a kerningu:

```
(LIGTABLE
   (LABEL O 40)
   (KRN C l R -0.277779)
   (KRN C L R -0.319446)
```

```
   (STOP)
...
 (LABEL O 41)
   (LIG O 140 O 74 )
   (STOP)
atd
)
```

První část zhruba říká, že znak s oktalovým kódem 40 má se znakem l kerning −0.277779 a se znakem L −0.319446. V druhé části se specifikuje, že znak na místě 41 následován znakem 140 dává znak 74.

Následují informace o jednotlivých znacích:

```
(CHARACTER O 54
   (CHARWD R 0.277779)
   (CHARHT R 0.105556)
   (CHARDP R 0.194445)
   (COMMENT
      (LIG O 54 O 376)
      )
```

Tedy znak 54 (čárka, opět oktalově) má definovánu šířku, výšku a hloubku (v komentáři uvedeno, že tvoří ligaturu sama se sebou, a to spodní uvozovky). Podrobněji např. [7].

PostScriptové metriky

I PostScriptové fonty mají své metriky. Nejvíce metrických informací je uloženo v souborech s koncovkou AFM (Adobe Font Metrics), které lze prohlížet přímo bez nějakých filtrů. Tedy do toho:

```
StartFontMetrics 2.0
Comment Copyright (c) 1984 Adobe Systems Incorporated.
Comment Creation Date:Mon Mar 31 17:10:33 PST 1986
FontName Times-Roman
EncodingScheme AdobeStandardEncoding
FullName Times Roman
FamilyName Times
Weight Roman
ItalicAngle 0.0
IsFixedPitch false
UnderlinePosition -109
```

```
UnderlineThickness 49
Version 001.000
Notice Times Roman is a trademark of Allied Corporation.
FontBBox -170 -217 1024 896
CapHeight 662
XHeight 448
Descender -217
Ascender 682
```

Opět hlavička s mnoha informacemi, jednotky jsou v PostScriptových (Didotových) bodech[1] (72 do palce) vynásobeno stem.

Následují informace o jednotlivých znacích:

```
StartCharMetrics 210
...
C 33 ; WX 333 ; N exclam ; B 109 -14 224 676 ;
...
C 102 ; WX 333 ; N f ; B 20 0 383 682 ; L i fi ; L l fl ;
C 174 ; WX 556 ; N fi ; B 33 0 521 678 ;
...
C -1 ; WX 722 ; N Aacute ; B 15 0 706 892 ;
...
EndCharMetrics
```

Po C je uvedeno číslo znaku (dekadicky v rozsahu 0–255). Pokud je tam -1, znamená to, že znak je kompozitní (složený z více znaků) a jeho definice bude uvedena později. Za WX je uveden posun sázecího bodu, tedy vlastně šířka znaku. Po N následuje jméno (Aacute znamená velké A s čárkou). Následuje BoundingBox, první dvě čísla udávají levý spodní roh, druhá dvě čísla pravý horní roh, nejprve x-ová osa, poté y-ová osa. Za položkou L je uvedeno, tvoří-li znak s nějaký jiným znakem ligatury, tedy f tvoří se znakem i ligaturu, která se jmenuje fi. Číslo znaku udává pozici v tzv. encoding vektoru, o kterém si povíme v kapitole o PFB a PFA souborech.

Následují kerningové informace:

```
StartKernData
StartTrackKern 3
Comment Light kerning
```

[1] Podle definice, 1 m = 2660 Didotových bodů (v TEXu se značí dd). Velikost PostScriptového bodu je jiná, v TEXu se značí bp, 1 in = 72 bp. Pozn. red.

```
TrackKern -1 14 0 72 -1.89
...
EndTrackKern
StartKernPairs 113
KPX A y -92
KPX A w -92
...
EndKernPairs
EndKernData
```

Jak vidno, tento oddíl sestává ze dvou částí: v první se definuje kerning takříkajíc plošný (bezkontextový). Při určování rozpalů se nehledí na jednotlivé znaky, ale mezi každé dva se vrazí mezera jisté velikosti závislá na TrackKernu a velikosti písma. Stručně řečeno, pro písma menší než 14 pt je kerning 0, pro písma větší než 72 pt je kerning -1.89, pro písma mezi se kerning spočítá lineární interpolací.

Druhá část se týká kontextového (párového) kerningu a říká: mezi písmeny A a y stáhni mezeru o 0.92 pt. Nutno podotknout, že zde může být uveden i vertikální kerning, pokud KPX nahradíme KPY, případně KP, pokud chceme užít obě posunutí.

Zbývá již jen popsat kompozity:

```
StartComposites 56
CC Zcaron 2 ; PCC Z 0 0 ; PCC caron 139 214 ;
...
EndComposites
EndFontMetrics
```

Tedy znak Zcaron sestává ze dvou komponent, první – Z – je položena na pozici 0 0, druhá – caron (háček) – je posunuta na pozici 139 214. Hloubavější zájemci o formát AFM nechť nahlédnou do [8].

Program `afm2tfm`

Z předchozího výkladu je zřejmé, že se obě metriky rámcově podobají. Jediný podstatný rozdíl (pomineme-li, že AFM-soubory mohou obsahovat ještě informace pro směr sazby a mnoho „jemnůstek“) jsou kompozitní znaky a fakt, že v TFM souboru může být nejvýše 256 znaků. Avšak TEX pro zhotovení (spočtení) sazby nijak nepotřebuje informaci o tom, jak jsou které znaky vytvořeny. Proto se tato informace uchovává odděleně v takzvaném virtuálním fontu, o kterém si povíme za chvíli.

Nikoho v tuto chvíli asi nepřekvapí, že lze z AFM souboru získat TFM soubor, a to pomocí programu `afm2tfm`. Program rozumí kerningům, ligaturám i kompozitním znakům.

Nejjednodušší, co můžeme udělat, je napsat:

```
afm2tfm Times-Roman rptmr
```

Program pak ze souboru Times-Roman.afm vybere základní znaky (t. j. ty které mají číslo 0–255) a vytvoří TFM soubor z těchto znaků, kerningy, ligatury a kompozity zanedbá. Tento font se pak nazývá *raw* (viz to r na začátku názvu rptmr).

Takový font je nám vlastně k ničemu, mnohem častěji použijeme volání

```
afm2tfm Times-Roman -v ptmr rptmr
```

a výsledkem kromě souboru `rtpmr.tfm` bude i soubor `ptmr.vpl`. Přípona VPL (virtual property list) znamená, že soubor je mixem TFM informací a informací virtuálního fontu. Je to tedy jakási obdoba AFM souboru (t. j. obsahuje metrické, kerningové i kompozitní informace). Jediná slabost VPL souboru spočívá v tom, že může obsahovat pouze 256 znaků, zatímco AFM soubor jich smí obsahovat libovolné množství (ale jen 256 jich má svoje číslo). Podívejme se nyní na soubor `ptmr.vpl`:

```
(VTITLE Created by afm2tfm ptmr0 -v ptmr)
(COMMENT Please edit that VTITLE if you edit this file)
(FAMILY TeX-rptmr)
(CODINGSCHEME TeX text + AdobeStandardEncoding)
(DESIGNSIZE R 10.0)
(DESIGNUNITS R 1000)
(COMMENT DESIGNSIZE (1 em) IS IN POINTS)
(COMMENT OTHER DIMENSIONS ARE MULTIPLES OF DESIGNSIZE/1000)
(CHECKSUM O 7575461244)
(FONTDIMEN
   (SPACE D 250)
   (STRETCH D 200)
   (SHRINK D 100)
   (XHEIGHT D 448)
   (QUAD D 1000)
   (EXTRASPACE D 111)
   )
```

```
(MAPFONT D 0
   (FONTNAME rptmr)
   (FONTCHECKSUM O 30202316533)
   )
...
```

Vidíme, že se podobá TFM souboru, pouze přibyly položky MAPFONT, jimiž specifikujeme, z kterých fontů se bude výsledný font skládat, a VTITLE.

Dále ligační schéma:

```
(LIGTABLE
   (LABEL O 40)
   (LIG C L O 350)
   (LIG C l O 370)
   (STOP)
   (LABEL O 41)
   (LIG O 140 O 16)
   (STOP)
   (LABEL O 47)
   (LIG O 47 O 272)
   (KRN O 47 R -74)
   (KRN C s R -55)
   (KRN C t R -18)
   (STOP)
...
```

Tedy nic nového.

A specifice znaků:

```
(CHARACTER O 15 (comment quotesingle)
   (CHARWD R 180)
   (CHARHT R 685)
   (MAP
      (SETCHAR O 251)
      )
   )
```

Vidíme, že na pozici 15 se namapoval znak 251 z fontu rptmr, který má dané rozměry (pokud změníme šířku znaku, můžeme tak vlastně vytvořit proložený font),

nebo:

```
(CHARACTER O 43 (comment numbersign)
   (CHARWD R 500)
   (CHARHT R 662))
```
znamená, že se na tuto pozici mapuje stejný znak (43), nebo konečně kompozity:
```
(CHARACTER O 202 (comment Aacute)
   (CHARWD R 722)
   (CHARHT R 892)
   (MAP
      (SETCHAR O 101)
      (MOVERIGHT R -528)
      (MOVEUP R 214)
      (SETCHAR O 302)
      )
)
```
Tedy na pozici 202 namapuj nejprve znak 101, pak se šoupni doprava (doleva), nahoru a namapuj znak 302. Možnosti virtuálního fontu jsou mnohem širší. Virtuální popis může obsahovat navíc i různé `specialy`, např. PostScriptové. Pěkný příklad je v [7, str. 75–79], za nahlédnutí stojí i [5].

Co tedy dál? Na soubor `ptmr.vpl` zavoláme program:

```
vpltovf ptmr.vpl
```

který nám vytvoří soubory `ptmr.tfm` a `ptmr.vf`. První potřebuje TEX pro sazbu, druhý (tzv. virtuální font) je vyžadován ovladačem pro zobrazení. Pokud ovladač nepodporuje virtuální fonty (např. dviwin pro Windows), pak si lze pomoci programem `dvicopy`, který dvi-soubor „odvirtualizuje“ (nahradí v něm kompozity skutečnými znaky).

Kódování výsledného fontu se ve spodní části kryje s kódováním cm-fontů (pokud to font umožňuje), ve vrchní části pak je doplněno původní kódování. Nejčastější změnou, kterou chceme s fontem udělat, je právě změna kódování (na kódování $\mathcal{CS}$-fontů). Program `afm2tfm` to umožňuje hned dvěma způsoby:

1. Překódování *raw*-fontu (tedy toho, na který odkazujeme z virtuálního fontu). Tato metoda se hodí hlavně v případě, že máme velký font t. j. má více než 256 skutečných (nekompozitních) znaků. Toto tvrzení se zdá poněkud ve sporu s předchozími prohlášeními, že v AFM souboru jich může být pouze 256. Takové obludy snadno

vznikají např. konverzí z TrueTypů, a možná že je novější verze AFM dokonce podporují. Pokud je nějaký znak, na který se chceme odkazovat, mimo rozsah 0–255, pak nezbývá, než ho posunout na jiné místo, neboť `afm2tfm` odmítne na takovýto znak napsat odkaz (nemluvě už o TEXu a ovladačích). Pokud pro zobrazení a tisk používáme pk-bitmapy generované programem `ps2pk`, musíme použít parametru `-E` pro překódování. Pokud chceme využít DVIPS a PFB font, pak do souboru `psfonts.map` napíšeme něco jako:

```
rptmr Times-Roman "CSencoding ReEncodeFont" <myenc.enc
```

viz kapitola o DVIPS. Dlužno podotknouti, že se mi toto nikdy nepodařilo[2], t. j. nikdy se mi nevygeneroval ten správný postscript. Volání programu `afm2tfm` je v tomto případě:

```
afm2tfm Times-Roman -p myenc.enc -v ptmr rptmr
```

2. Druhá metoda nechává *raw*-font na pokoji (tedy vlastně počítá s tím, že má nejvýše 256 znaků) a mění VPL, potažmo VF font, tedy znaky pouze přemapuje. Lze použít i s pk-fonty, i s DVIPS poměrně bez problémů. Její volání je

   ```
   afm2tfm Times-Roman -t myenc.enc -v ptmr rptmr
   ```

Encoding

v obou předchozích metodách jsme potřebovali soubor myenc.enc, který specifikuje naše kódování. Povězme si nyní něco málo o něm.

Vlastní soubor sestává ze dvou částí (nepočítaje komentáře). První je vlastně Encoding vektor (PostScriptový array) a má přesně 256 (jaký to div) položek. Číslo položky je specifikováno jejím pořadím. Tedy:

```
/CSencoding [ /Gamma /Delta /Theta /Lambda /Xi /Pi /Sigma
/Upsilon1 /Phi /Psi /Omega /ff /fi /fl /ffi /ffl /dotlessi
/dotlessj /grave /acute /caron /breve /macron /ring
/cedilla /germandbls /ae /oe /oslash /AE /OE /Oslash
/.notdef /exclam /quotedblright /numbersign /dollar
/percent /ampersand /quoteright /parenleft /parenright
```

[2] Redakčním zásahem byl vyhozen konec řádku ve výše uvedeném příkladu, kde bylo napsáno `<ptmr.pfb`, a to hned ze dvou důvodů. Jednak se tak dlouhý řádek nevešel to tiskového zrcadla, a navíc právě zde je příčina neúspěchu. Font Times-Roman je totiž rezidentní ve všech PostScriptových zařízeních, takže soubor `ptmr.pfb` na disku obvykle nenajdete. Pozn. red.

```
/asterisk /plus /comma /hyphen /period /slash /zero /one
/two /three /four /five /six /seven /eight /nine /colon
/semicolon /exclamdown /equal /questiondown /question /at
/A /B /C /D /E /F /G /H /I /J /K /L /M /N /O /P /Q /R /S /T
/U /V /W /X /Y /Z /bracketleft /quotedblleft /bracketright
/circumflex /dotaccent /quoteleft /a /b /c /d /e /f /g /h
/i /j /k /l /m /n /o /p /q /r /s /t /u /v /w /x /y /z
/endash /emdash /hungarumlaut /tilde /dieresis /.notdef
/.notdef /.notdef /.notdef /.notdef /.notdef /.notdef
/.notdef /.notdef /.notdef /.notdef /.notdef /.notdef
/perthousand /.notdef /.notdef /.notdef /.notdef /.notdef
/.notdef /.notdef /.notdef /.notdef /.notdef /Agrave
/.notdef /.notdef /.notdef /hyphen /ogonek /guillemotleft
/guillemotright /.notdef /Acedilla /breve /Lslash /currency
/Lcaron /Sacute /section /dieresis /Scaron /Sogonek /Tcaron
/Zacute /.notdef /Zcaron /Zdotaccent /ring /acedilla
/cedilla /lslash /acute /lcaron /sacute /caron /agrave
/scaron /sogonek /tcaron /zacute /hungarumlaut /zcaron
/zdotaccent /Racute /Aacute /Acircumflex /Abreve /Adieresis
/Lacute /Cacute /Ccedilla /Ccaron /Eacute /Eogonek
/Edieresis /Ecaron /Iacute /Icircumflex /Dcaron /Dslash
/Ndotaccent /Ncaron /Oacute /Ocircumflex /Ohungarumlaut
/Odieresis /multiply /Rcaron /Uring /Uacute /Uhungarumlaut
/Udieresis /Yacute /Togonek /germandbls /racute /aacute
/acircumflex /abreve /adieresis /lacute /cacute /ccedilla
/ccaron /eacute /eogonek /edieresis /ecaron /iacute
/icircumflex /dcaron /dslash /ndotaccent /ncaron /oacute
/ocircumflex /ohungarumlaut /odieresis /divide /rcaron
/uring /uacute /uhungarumlaut /udieresis /yacute
/quotedblbase /quotedblleft ]
```

Vidíme, že znaky jsou uvedeny pod svými adobe názvy. Pokud na daném místě není namapován žádný znak, stojí tam napsáno: `/.notdef` Všimněme si, že kódování má své jméno: `CSencoding`, které musí být unikátní (zvláště pokud užíváme více kódování fontů v jednom dokumentu). Pokud není, mohou později nastat potíže.

Druhá část specifikuje ligatury:

```
% LIGKERN hyphen hyphen =: endash ; endash hyphen =: emdash ;
```

```
% LIGKERN quoteleft quoteleft =: quotedblleft ;
% LIGKERN quoteright quoteright =: quotedblright ;
% LIGKERN exclamdown exclamdown =: guillemotleft ;
% frenchdblquotes
% LIGKERN questiondown questiondown =: guillemotright ;
% csquoteleft
% LIGKERN comma comma =: quotedblbase;
% LIGKERN space {} * ; * {} space ; zero {} * ; * {} zero ;
% LIGKERN one {} * ; * {} one ; two {} * ; * {} two ;
% LIGKERN three {} * ; * {} three ; four {} * ; * {} four ;
% LIGKERN five {} * ; * {} five ; six {} * ; * {} six ;
% LIGKERN seven {} * ; * {} seven ; eight {} * ; * {} eight ;
% LIGKERN nine {} * ; * {} nine ;
% LIGKERN question {} quoteleft ; exclam {} quoteleft ;
```

První řádek říká: dva hypheny dají jeden endash, endash a hyphen dá emdash. Příkazy typu `one {} *` říkají: odstraň kerning mezi jedničkou a čímkoliv (např. `* {} *` odstraní všechny kerningy, notace `a{}b` je podobná TEXovské). Více (např. koncové ligatury) v manuálu k DVIPS.

Program `afm2tfm` má ještě několik jiných možností jako například: natahování (zužování) znaků pomocí parametru `-e`, vytváření kapitálek pomocí `-c`, nebo skloněného fontu pomocí `-s`, jimiž se však nebudeme na tomto místě zabývat. Dlužno podotknouti, že původně měl obsahovat i vytvoření prokládaného fontu, od této myšlenky se však upustilo pro potíže s ligaturou fi (a ostatními). Nebylo jasné, má-li se trhat, nebo ponechávat. Více manuál DVIPS: A TEX Driver, nebo článeček [1].

Outliny a bitmapy

Takže už umíme připravit metriky a zbývá naučit driver zobrazovat správně znaky. Nejprve tedy trocha teorie:

PFB, PFA a ostatní

Existuje stará pověra:

> *PostScriptové fonty jsou dvou typů Type1 a Type3, první mají příponu PFA, druhé PFB.*

Jak to tedy je? Za prvé existují asi čtyři druhy PostScriptových fontů Type1, Type3, Type4 a Type5, z niž druhé dva se od prvního liší pouze formátem (Type4 je formát pro uložení na disku, Type5 pro uložení v tiskárně). Ovšem přípony PFA a PFB se obě týkají formátu Type1. První říká, že jde o ASCII kódování, druhá, že jde o kódování binární (úsporné).

Co jsou tedy vlastně Type1 a Type3 zač:

Type1 Vektorový popis fontu, který neobsahuje „plný“ PostScript. Obsahuje pouze operace pro vykreslení obrysu „lineto, curveto...“ plus navíc hintovací operace opravující špatný vzhled bitmap v nižších rozlišeních (potažmo v malých velikostech). Protože jejich jazyk je slabý, lze je vykreslit poměrně rychle. O to se stará ATM (Adobe Type Manager), který je rastruje a výsledné bitmapy posílá aplikaci.

Type3 Využívá „plný“ PostScript. Je vhodný zejména pro bitmapové fonty nebo různě divoké výplně fontů (třeba fraktálové, máte-li dost času na tisknutí). Neobsahuje hinty (to není záležitost PostScriptu) a ATM jim nerozumí.

Asi už vám došlo, že dál budeme sjíždět pouze Type1. Nejprve je trochu rozpitváme, abychom lépe porozuměli tomu, co se děje pod pokličkou. Použijeme sadu volně dostupných operačních nástrojů T1UTILS. Programem T1DISASM převedeme font do čitelné podoby. Trochu se podívejme na výsledek. Nejprve hlavička:

```
%!PS-AdobeFont-1.0
%%CreationDate: Sat Feb 04 01:32:58 1995
%%VMusage: 27904 27904
% Created with FontMonger Copyright (c) 1991-2 ...
11 dict begin
/FontInfo 9 dict dup begin
/version (001.000) readonly def
/Notice () readonly def
/FullName (HelveticaInseratE  Medium) readonly def
/FamilyName (HelveticaInseratE-) readonly def
/Weight (Medium) readonly def
/ItalicAngle 0 def
/isFixedPitch false def
/UnderlinePosition -100 def
```

```
/UnderlineThickness 50 def
end readonly def
/FontName /HelveticaInseratEMedium def
```

Nejedná se vlastně o hlavičku, ale o počáteční definice. Následuje Encoding Vector (zkráceno):

```
/Encoding 256 array
0 1 255{1 index exch /.notdef put}for
dup 32 /space put
dup 33 /exclam put
dup 34 /quotedbl put
dup 35 /numbersign put
dup 36 /dollar put
dup 37 /percent put
dup 38 /ampersand put
dup 39 /quotesingle put
dup 40 /parenleft put
dup 41 /parenright put
dup 42 /asterisk put
dup 43 /plus put
...
readonly def
```

Způsobů zadávání Encodingu je celá řada. Nelekněte se proto, když narazíte na jiný složitější (podívejte se např. na font Utopia). Zde vidíme, že celý array se nejprve vynuluje (vynotdefuje), a pak se teprve plní. Porovnejte jej s kódováním v příslušném AFM souboru, měly by být stejné.

Následují další obecné definice a nakonec jednotlivé znaky:

```
/space {       0 250 hsbw
        endchar
        } ND
/exclam {      84 334 hsbw
        0 166 vstem
        0 184 hstem
        779 20 hstem
        799 vmoveto
        -312 vlineto
        38 -256 rlineto
```

```
90 hlineto
38 256 rlineto
312 vlineto
closepath
-799 vmoveto
184 vlineto
-166 hlineto
-184 vlineto
closepath
endchar
} ND
```

Pro neznalce PostScriptu podotýkám, že vše je v obrácené notaci, nejprve operandy, pak operátory. Řádky končící na hstem nebo vstem obsahují právě hintovací informace (z jejich výskytu lze usoudit kvalitu fontu).

Jak to použít v TEXu

Takže máme AFM a PFB soubor a chceme vytvořit fungující (dejme tomu český) font. Předpokládejme na chvíli, že již obsahuje české znaky. Dále máme soubor `csenc.enc`, kde je specifikováno české kódování. Pro začátek zvolíme druhou metodu získání TFM souboru:

```
afm2tfm Times-Roman -t csenc.enc -v ptmr8z rptmr
```

Někoho možná napadne, proč zrovna ptmr8z a rptmr. Je to otázka konvence, a to konkrétně „Font Naming Conventions“, jejímž autorem je Karl Berry, a více se lze o ní dočíst v manuálu k DVIPS. Stručně: skládá se ze jména písmolijny, názvu, duktu a asi tří dalších položek (p znamená PostScript (adobe), tm – Times, r – Roman, 8z je kódování). Ze CTANu lze stáhnout (stále se prodlužující) seznam písem s jejich správnými (konvenčními) jmény. Je dobré se touto konvencí řídit, by nedošlo k zmatení jazyků.

Pokud v tomto okamžiku nastane chyba (při volání `afm2tfm` nebo následném `vpltovf`), může to znamenat, že font má více než 256 základních znaků, a my ve virtuálním fontu odkazujeme na nějaký vyšší znak. (Pozná se snadno z toho, co programy hlásí.) Situace se dá řešit několika způsoby:

1. Předělat AFM a PFB soubory k obrazu svému (někde jsem četl hlášku: „God saves the T1UTILS“). Pozor na práva, někteří au-

toři jsou hákliví na své dílo (byť jsou jeho kvality pochybené). Lze k tomu s uspěchem použít i komerční programy FontMonger a FontoGrapher.

2. Využít první metodu a překódovat *raw*-font a doufat, že konečně někdo rozchodí i ReEncodeFont. Jinak jsme odkázáni na `ps2pk`.

Některé verze `vptovf` se špatně snášejí s některými verzemi `afm2tfm`, zejména si nerozumí v koncích řádků. Lékem většinou bývá filtr `unix2dos` (nebo prosté načtení do některého z editorů a následné uložení). Občas vygeneruje `afm2tfm` dokonce špatný VPL soubor (pozná se to však až při prohlížení), a je potřeba celou operaci provést znovu. (Neznám genealogii této chyby, ale v poslední verzi programu jsem se s ní již nesetkal.)

Konečně přichází fáze testování. Já osobně používám program `ps2pk` k vygenerování bitmap, protože funguje pro obě varianty překódování fontu.

Ps2pk

Je Public Domain balík založený na komerčním rastrovacím programu, který IBM darovala X-consortiu. Umí tedy rastrovat (včetně hintingů) Type1 fonty do pk bitmap v různých rozlišeních a velikostech. Navíc umí překódování fontu, roztažení a sklonění fontu (dalo by se říci, že co dělá `afm2tfm` pro AFM soubory, dělá `ps2pk` pro PFB soubory). V nové verzi (určitě si ji stáhněte) je navíc spojen s programem, udržujícím resource databázi, a obdobou MFjobu, generující chybějící fonty (a to jak METAFONTové, tak Type1). Pokud si celý balík správně nainstalujete a udržujete, nemusíte se nadosmrti starat, co je jaký font, a jak je překódován. Pokud vývoj půjde tímto směrem, zřejmě se `ps2pk` stane jakýmsi ATM pro TEX.

Program má i své chyby. Například všechny znaky jsou generovány o jeden dot výše, což se projeví, pokud spojujete Type1 fonty např. s cm fonty, ale při rozlišení 300 dpi je na to potřeba už pořádná lupa. Druhý problém je s hintingy. Vyskytnou se fonty, které `ps2pk` vyrastruje příšerně na rozdíl od PostSriptové tiskárny, a navíc je to viditelné i ve vyšších rozlišeních. Smůla.

Více tedy manuál k `ps2pk`. Obecná poučka zní: výsledné pk-fonty musí mít stejné jméno a kódování jako *raw*-fonty (podobně je tomu s rozšířením a skloněním). Takže pokud jste generovali TFM soubor pomocí:

```
afm2tfm Times-Roman -p csenc.enc -v ptmr8z rptmr
```
voláme pak `ps2pk`
```
ps2pk -E csenc.enc Times-Roman rptmr.pk
```
V případě
```
afm2tfm Times-Roman.pfb -t csenc.enc -v ptmr8z rptmr
```
použijeme
```
ps2pk Times-Roman.pfb rptmr.pk
```
Umístíme nyní TFM, VF, a PK soubory na správná místa (aby je ovladač našel) a zkusíme si vytisknout tabulku fontu. Používám k tomu makro `testfont`. Výslednou tabulku porovnáme s tabulkou fontu csr10. Pokud je vše O.K., můžeme si blahopřát a přejít k dalšímu kroku. Pokud pouze chybí některé znaky, podíváme se do AFM souboru, zda jsou ve fontu obsaženy. Pokud skutečně nejsou, nebo pouze chybí kompozitní popisy (jako třeba ve fontu Utopia), můžeme s výhodou použít program a2ac Petra Olšáka. Někdy se stane, že v AFM souboru jsou znaky označeny jako kompozity, chybí v nich jejich popis (CC položka), ale v PFB souboru jsou obsaženy jejich obrysy. Pak stačí do AFM souboru opsat číslo z PFB. Při té příležitosti není na škodu zkontrolovat, jestli obě kódování souhlasí. Rovněž stojí za to zkontrolovat, zdali mají všechny znaky uvedené v Encoding vektoru odpovídající popisy v PFB souboru, případně, jsou-li všechny obrysy uvedeny v Encoding vektoru (zbytečně bychom tak přišli o nějaké znaky).

Nejhorší případ nastane, když znaky nejsou na správných místech. Pak je třeba nejprve pro jistotu zkontrolovat náš kódovací soubor a není-li chyba v něm, pak vězte, že máte co do činění s chybou nejstrašnější, bohužel častou. Někteří samozvaní počešťovatelé nejen že vyházejí z fontu ligatury (tomu jsme již přivykli), leč se nenamáhají dát obrysům jejich pravá jména (dlouhé A pak např. najdeme pod názvem iacute). Řešení je vcelku bolestivé. Zjistíme správnou permutaci buď pomocí komerčního software (používám FontoGrapher) nebo pomocí GhostScriptu nebo lze využít `pkedit` a námi vygenerovanou bitmapu. Pak např. awkovským skriptem za tímto účelem zhotoveným proženeme „vadné“ soubory a zjednáme nápravu. Celý postup opakujeme.

Většinou zbývá doladit několik jemnůstek. Pokud například víme, že ve fontu je obsažena ligatura fi, ale nám se ne a ne zjevit, je to způsobeno většinou absencí příznaku `L i fi` v AFM souboru a není těžké jej doplnit.

DVIPS

Konečně jsme tedy font odladili a zbývá jej rozchodit s DVIPS. Pokud jsme nepoužívali překódování *raw*-fontu, je to relativně snadné. Doplníme do souboru `psfonts.map` řádek:

```
rptmr Times-Roman <Times-Roman.pfb
```

První jméno je *raw*-font, druhé je FontName z AFM souboru a třetí je PFB soubor. Umístíme pak `Times-Roman.pfb` na místo, kde ho DVIPS najde a pokusíme se nyní vytisknout tabulku fontu pomocí DVIPS. Pokud vše proběhne v pořádku (DVIPS nenahlásí žádnou chybu a opravdu font natáhne), ale tiskárna vytiskne vše fontem Courier (psací stroj), podíváme se ještě, zdali `FontName` je stejné v AFM i PFB souboru. Pokud ne, sjednotíme je (nejlépe tak, že to opravíme v souboru AFM a `psfont.map`, je to nejméně předělávání a nemusíme znova spouštět `afm2tfm`)

Pokud jste použili metodu s překódováním *raw*-fontu, v manuálu DVIPS je psáno, že funguje vložení následujícího řádku do `psfonts.map`:

```
rptmr Times-Roman "CSencoding ReEncodeFont" <csenc.enc
```

kde název `CSencoding` musí být nachlup stejný, jaký je v souboru `csenc.enc`. Podotýkám, že mně to zatím nechodí[3], snad budete šťastnější (v poslední distribuci DVIPS je celý direktorář věnovaný tomuto problému).

Varování

I když máte font legálně koupený, nelze věřit takřka ničemu. Sám jsem opravoval font, kde se neshodovalo: FontName v AFM a PFB souboru, některé znaky byly uvedeny v AFM souboru jako kompozitní, byť jejich obrysy byly obsaženy v PFB souboru (a pochopitelně jejich kompozitní popisy v AFM souboru chyběly), a některé jiné obrysy obsažené v PFB souboru nebyly ani v jednom Encoding vektoru (jednalo se o písmena ť a ď, čili nic zanedbatelného). Nejpodivnější bylo, že se celý font choval pod Windows správně[4].

[3] Viz poznámku na str. 258 nebo též kapitolu 5.3 v dokumentaci programu DVIPS. Pozn. red.

[4] Vysvětlení je nasnadě. ATM pro Windows nepoužívá AFM soubory. Část metrických informací (BoundingBox) čerpá přímo z PFB souboru, druhou část (kerning) má uloženu v PFM souboru (speciální binární tvar). Na Encoding se zřejmě neohlíží vůbec.

TrueTypy

Alternativou k Type1 jsou TrueTypy firem Apple a Microsoft. Jejich hlavní výhodou oproti PostScriptovým fontům má být automatické odstraňování „dropoutů“, tedy přerušení čáry v nižších rozlišeních, bohatší možnosti hintování a rychlejší vykreslování kvadratických splajnů (Type1 užívají kubické). Projeví se to zejména na obrazovce. Obecně lze bohužel říci, že TrueTypy jsou horší kvality nežli Type1, a to z několika důvodů:

1. Většinou vznikly špatnou konverzí z Type1. Konverze obrysů je nepoměrně jednoduší (byť netriviální), než konverze hintů.
2. Software na výrobu dobře ohintovaných TrueTypů je poměrně drahá záležitost (rozhodně to nezvládne ani FontoGrapher ani FontStudio) a hinty samotné jsou nezřídka ruční práce. V době psaní tohoto článku jsem neznal žádný dostupný WYSIWYG software (pro Windows) produkující skutečné TrueTypy. Vždy se jednalo o kubické splajny posléze konvertované na kvadratické, které ovšem program tajil (nezobrazoval).
3. Formát TrueTypů se zcela macešsky chová k ligaturám. To neznamená, že tam nejsou, ale pouze chybí informace, které páry tvoří jaké ligatury. Sázecí software může pouze tipovat.

Zatím jedinou možností, jak implantovat TrueTypy do TEXu, je jejich konverze do Type1[5]. Naštěstí konverze kvadratických splajnů na kubické dopadá veskrze lépe než obrácená, nemluvě o tom, že některé softwary umí zrestaurovat původní (kubické) obrysy, vznikl-li font např. konverzí z Type1.

Žel neznám jiný než komerční konverzní software.

FontMonger je program na tvorbu vektorových fontů zvládající i konverzi mezi mnoha formáty. Verze, kterou jsem užíval, měla potíže s kódováním. Důsledkem bylo zrušení některých znaků ve fontu během editace či konverze, čili nešla použít na většinu počeštěných fontů.

FontoGrapher neumí konvertovat mezi tolika formáty jako FontMonger, ale na rozdíl od něho se nezalekne ani skurilních kódování

[5] Existuje sice komerční verze TEXu TrueTEX, ale nepředpokládám její hojný domácí výskyt.*

*Pracuje se na konvertoru TTF2AFM (nekomerčním), který umožní použití TrueType fontů v TEX2PDF. Pozn. red.

ani více znaků ve fontu než 256. Rovněž zvládá i vektorizaci (malých) bitmap, což jej povyšuje na vhodný nástroj k domácí výrobě Type1 (resp. TrueType) fontů. Nezanedbatelná je i (vypnutelná) autohintovací funkce.

Hodně štěstí

Literatura

[1] Petr Sojka: Virtuální fonty, accents a přátelé. Zpravodaj Československého sdružení uživatelů TEXu, 1994, # 2, str. 56–69.
(`ftp://ftp.muni.cz/pub/tex/local/cstug/sojka/aboutacc/`)

[2] Joachim Schrod: Komponenty TEXu. Zpravodaj Československého sdružení uživatelů TEXu, 1993, # 1, str. 1–10.
(`ftp://ftp.muni.cz/pub/tex/CTAN/documentation/components-of-TeX/`)

[3] Tomáš Mráz: PostScriptová písma v TEXu. Zpravodaj Československého sdružení uživatelů TEXu, 1994, # 2, str. 70–76.

[4] Petr Olšák: Program a2ac – manipulace s fontem na úrovni PostScriptu. Zpravodaj Československého sdružení uživatelů TEXu, 1996, # 1, str. 23–30.
(`ftp://math.feld.cvut.cz/pub/cstex/msdos/cspsfont.zip`)

[5] Donald Knuth: Virtual fonts, a more fun for grand wizards. TUGboat, 1990, # 1, str. 13–23.
(`ftp://ftp.muni.cz/pub/tex/CTAN/documentation/virtual-fonts.knuth`)

[6] Pavel Herout: PostScriptové fonty pro ty, co o nich moc nevědí. Zpravodaj Československého sdružení uživatelů TEXu, 1996, # 1, str. 43–64.

[7] Petr Olšák: Typografický systém TEX. Československé sdružení uživatelů TEXu, 1995.

[8] Adobe Font Metrics File Format Specification Version 4.0. Adobe Systems Incorporated, February 1992.

Arnošt Štědrý
`arnost@uivt.cas.cz`

LaTeXová kuchařka/2

Zdeněk Wagner

Otvíráme druhý díl LaTeXové kuchařky. Minule jsme se zaměřili na základy, o nichž se již dále zmiňovat nebudeme. Předpokládáme, že čtenář tohoto dílu již ví, jak se má zachovat, když se mu v dokumentu objeví chyba `Invalid use of \spacefactor`.

Nyní se již začneme zabývat metodami, s jejichž pomocí budeme zasahovat do vzhledu dokumentu. Povíme si o problematice, která v plainu není řešena. Zatímco uživatel plainu musí nadefinovat vše sám, LaTeXistovi stačí pouhá úprava několika málo maker.

Abychom zdůraznili, že se jedná o pokračování, navážeme číslování kapitol. Trochu předběhneme výklad a ukážeme, jak to bylo provedeno. Minulý díl končil kapitolou 7. Proto je na začátku tohoto dokumentu:

```
\setcounter{section}{7}
```

Příkaz `\section` zvětší tento čítač o jednotku a příslušnou hodnotu vytiskne. Tím jsme dosáhli, že následující kapitola má číslo 8.

8. Změna písma

Tento text je zařazen jako reakce na problémy s CSTeXem. V počátku byly do tohoto balíku zařazeny virtuální skripty počeštěných PostScriptových fontů včetně `.fd` souborů pro LaTeX. Omylem však CSTeX neobsahoval balík PSNFSS. Uživatelé pak došli k závěru, že PostScriptové fonty nelze v LaTeXu použít. To byla ovšem pravda pouze částečná. Nestačilo napsat `\usepackage{times}`, neboť příslušný balík nebyl instalován. Řešením byl povel:

```
\fontfamily{ptm}\selectfont
```

Nyní si vysvětlíme, jak a proč výše uvedený příkaz funguje.

8.1. Charakteristiky písma

Typografie rozeznává kromě velikosti tři další charakteristiky písma. LaTeX přidává jeden atribut navíc, a o tom si povíme nejdříve. Snad

každý uživatel TEXu se ve svých začátcích pokusí napsat v textu znaky < a >. Výsledkem je ovšem ¡ a ¿. V matematických vzorcích mu správně fungovalo < i >. Uživatel je tedy zmaten. Na vině je právě ten čtvrtý atribut, to je kódování fontu.

Představte si, že by klávesy na vašem počítači byly jednoznačně očíslovány, například pořadovými čísly. Takovému očíslování můžeme říkat kódování. Vskutku má každá klávesa interně přiřazeno nějaké číslo – kód. Při stisku klávesy je tento kód vyslán do počítače a příslušné vstupní obvody tak poznají, kterou klávesu uživatel zmáčkl. Pokud se má nějaký znak zobrazit na obrazovce či vytisknout na tiskárně, musí se opět vyslat pořadové číslo tohoto znaku v příslušném fontu. Přitom počítač kóduje znaky jinak než klávesnice a kód znaků ve fontu může také být zcela odlišný. Navíc je kódování různých fontů obecně různé. Zpracování dokumentu v (LA)TEXu[1] tedy zahrnuje několik konverzí. Nejprve operační systém převede kód klávesy na kód, kterému rozumí. To se odehrává zejména při psaní zdrojového textu editorem. TEX pak při čtení souboru konvertuje znaky do svého vnitřního kódování. Před tiskem opět provádí konverzi podle kódování použitého fontu. Pokud mají všechny fonty stejné kódování, můžeme TEX naučit pracovat přímo v kódování fontu. Druhá konverze tím odpadne. Pokud používáme pouze sedmibitový ASCII kód nebo máme na úrovni operačního systému stejný kód, jaký mají naše fonty (to je případ UNIXu a kódování IL2), nepotřebujeme konverzi žádnou.

Většina uživatelů si zřejmě vystačí s jediným kódováním textu. Připusťme však, že musíme do textu propašovat větu: „В системе LATEX возможно писать по русски.“ Je zřejmé, že азбука má některé znaky odlišné, takže i jejich umístění ve fontu (tj. kódování) bude jiné. Uvedený příklad lze vytisknout i beze změny kódování, ale totéž neplatí o fontech matematických. Zkuste si například vysázet nějaký text (nejlépe anglický) fontem `cmex10`. Ve výsledku asi původní text nepoznáte. Proto se matematické výrazy zapisují speciálními makry a konverzi do kódu fontu provádí TEX během jejich expanze.

Původně byl TEX vytvořen pro angličtinu a používal tedy pouze sedmibitový kód. Písmena s diakritikou v něm nebyla obsažena. Proto se „ř“ muselo skládat ze samostatného háčku a písmene „r“. Toto sedmibitové

[1] Tímto novotvarem autor vyjadřuje, že příslušné tvrzení platí jak pro plain TEX tak pro LATEX.

kódování se nyní označuje OT1. Později byl vytvořen osmibitový kód obsahující všechny potřebné akcentované znaky. Autoři LaTeXu očekávají, že jeho kódování, označované T1, se stane standardem. Rozložení znaků v kódech OT1 a T1 je dosti odlišné a pro matematickou sazbu je kód OT1 potřebný vždy. To brání zavedení nových fontů na staříčkých počítačích s malým diskem. Pro českou a slovenskou sazbu byl tedy vytvořen jiný font, který jednak šetří prostor na disku, a navíc má diakritická znaménka vytvořena podle národních typografických zvyklostí. Označuje se IL2 a je konzervativním rozšířením původního kódu OT1.

Představme si nyní, co se stane, když chceme vytisknout „ř“. Uvažujme nejsložitější případ, kdy chceme v jednom dokumentu použít několik různých fontů s odlišným kódováním a zpracováváme dokumenty psané s různým kódováním češtiny. Znak vyjadřující „ř“ v daném vstupním kódování musí být aktivní a definovaný tak, aby se expandoval na `\v{r}`. Tuto sekvenci již v zásadě můžeme použít k tisku „ř“, neboť příkaz `\v` je definován v každém textovém kódu. Například v OT1 bude konečným výsledkem expanze `\accent20r`. Primitiv `\accent` ovšem potlačí dělení slova. Pokud tedy příslušný font má znak „ř“, provede se odpovídající konverze. Dokument lze díky těmto vlastnostem zpracovat i na instalaci, která má pouze fonty s kódováním OT1. Výsledek sice bude vzhledem k potlačení dělení slov významně horší, nicméně pro hrubé čtení postačí. Pokud by tento složitý rys, a to jsme postup konverze popsali velmi stručně a nepřesně, nebyl implementován, nedal by se dokument na odlišné instalaci vytisknout vůbec (nebo až po mnoha zásazích do textu).

Problematika kódování je téměř vyčerpána, ale stále jsme nevysvětlili problém nastíněný v prvním odstavci této podkapitoly. Všimněte si rozdílu mezi < a $<$. První znak je v písmu simulujícím psací stroj, zatímco druhý znak je matematický. Znaky ¡ a ¿ pocházejí z obvyklého patkového písma Computer Modern, které ovšem znaky $<$ a $>$ neobsahuje. OT1 totiž není kódováním v pravém slova smyslu, neboť umístění znaků závisí i na volbě rodiny a řezu písma. Písmo psacího stroje má < na místě, kde základní proporcionální písmo obsahuje ¡. Při změně řezu na kurzívu se ze znaku \$ stane £. Tyto problémy dědí i kódování IL2.

O dalších atributech písma se zmíníme jen zkratkovitě. Nechceme zde suplovat učebnice typografie. Spíše uvedeme názvy charakteristik ve spojení s odpovídajícími LaTeXovými příkazy.

O rodině písma (`\fontfamily`) jsme se již zmínili. Dalšími charakteristikami jsou řez (`\fontshape`) a duktus (`\fontseries`).

8.2. Příkazy pro výběr písma

Řekli jsme si, že každé písmo je charakterizováno kódováním, rodinou, duktem, řezem a stupněm (velikostí). Všechny tyto charakteristiky budeme chtít ve svých dokumentech měnit nezávisle.

Výběr písma je implementován v plain TEXu i ve všech dalších formátech. Změna charakteristik však není zcela nezávislá.

Starý LATEX 2.09 poskytoval příkazy pro několik zvolených fontů. Přitom příkaz pro změnu velikosti písma současně nastavil základní rodinu, řez i duktus. Představte si, že jste měli větu:

Chceme ve větě napsat jedno slovo velké.

Nyní se rozhodneme převést celou větu do kurzívy. Výsledek nás překvapí. Uprostřed věty kurzíva nebude:

Chceme ve větě napsat jedno slovo *velké.*

Navíc nebyla zařazena ani kurzívní korekce. Při takové změně tedy bylo nutno projít celý text a na mnoha místech ručně dopisovat změny fontů. Pokud se nám přece jen kurzíva nelíbila a chtěli jsme obnovit původní stav, museli jsme opět změnit text na mnoha místech. Proto již pro LATEX 2.09 vytvořil Frank Mittelbach „New Font Selection Scheme“ (NFSS) umožňující ortogonální změny jednotlivých atributů. LATEX 2.09 ještě nerozeznával kódování fontů. To je implementováno až v tzv. NFSS2, které obsahuje LATEX 2_ε. Nyní tedy {`\it Chceme ve větě napsat {\Large jedno slovo} velké.`} vysází:

Chceme ve větě napsat *jedno slovo* *velké.*

Uvedený příklad se může zdát příliš vyumělkovaný. Je pravda, že typografická kvalita tohoto příkladu je dosti bídná. Představte si však, že máte delší text, kde jsou nadpisy vysázeny větším písmem, případně písmem tučným. Po provedení korektury jsme požádáni, abychom text přesadili například do švabachu. Dáme tedy za `\begin{document}` příkaz pro změnu fontu. Použijeme-li původní LATEX 2.09, budou větší a tučné nadpisy v Computer Modern Roman vypadat zvlášť ohavně. Musíme tedy ještě předefinovat formátování nadpisů – a v budoucnu to budeme muset opakovat pro každý jiný font. LATEX 2_ε nám tuto práci ušetří.

Za zmínku stojí i nová možnost. Díky ortogonálnímu výběru lze kombinací `\bfseries` a `\itshape` psát ***tučnou kurzívou***, což dřívější LaTeX nepřipouštěl bez zavedení nového fontu pomocí maker, jež nebyla ve zdrojovém kódu dokumentována. Aby se předešlo nejednoznačnostem při zpracování starých souborů, mají jednoduché příkazy `\bf`, `\it` a podobné týž význam jako v původním LaTeXu 2.09.

V minulé části jsme již uvedli, že máme makra deklarativní a příkazová. Příkazové makro, např. `\textit`, se hodí pro vysázení kratší části textu v kurzívě. Příkazová makra se automaticky starají o vkládání kurzívních korekcí. Deklarativní makro, např. `\itshape`, mění příslušný atribut uvnitř skupiny, v níž je použito. Hodí se zejména v uživatelských definicích dalších maker. Kurzívní korekce je nutno zadat explicitně.

Kurzívní korekce se vkládá při přechodu mezi skloněným a stojatým písmem. Nevypadalo by ovšem dobře, kdyby se korekce vložila před čárku, tečku, případně jiný podobný znak. Při automatickém vkládání se tedy kurzívní korekce vynechá v případě, že za ní následuje znak uvedený v `\nocorrlist`. Standardní definice tohoto makra je:

```
\newcommand{\nocorrlist}{,.}
```

Uživatel může toto makro předefinovat. Z hlediska rychlosti je vhodné, aby jednotlivé znaky byly uvedeny v pořadí četnosti výskytu v dokumentu (nejčastější na prvním místě). Kromě této globální změny lze potlačit vložení kurzívní korekce jednotlivě uvedením makra `\nocorr` na odpovídající straně textu v příkazovém makru pro změnu písma.

Atributy písma jsou voleny makry uvedenými v tabulce 1. Skutečný význam jednotlivých příkazů lze změnit na uživatelské úrovni. Každé makro totiž volá jistý „hook“, jehož definice může být změněna. Seznam těchto vnitřních maker včetně jejich hodnot je uveden v tabulce 2.

Makra z tabulky 1 tedy provádějí složitější činnost. Například `\bfseries` je expandováno na `\fontseries{\bfdefault}\selectfont`. Po změně makra `\bfdefault` zvolí tudíž `\bfseries` jiný duktus.

Některá písma mohou obsahovat rodiny, dukty či řezy, pro něž nejsou definována makra. Musíme tedy použít příkazy nižší úrovně. Pro změnu rodiny, duktu a řezu máme tedy postupně makra `\fontfamily`, `\fontseries` a `\fontshape`. Kódování se změní výhradně makrem `\fontencoding`. Tato makra pouze sdělí systému NFSS, že se nějaký atribut bude měnit, ale neprovedou vlastní výběr písma. Změna se provede až příkazem `\selectfont`. Zkuste si ve svém LaTeXu následující příkazy:

Tabulka 1: Makra pro změnu písma

Příkaz	*Deklarace*	*Význam*
`\textrm{...}`	`{\rmfamily...}`	Patkové písmo
`\textsf{...}`	`{\sffamily...}`	Bezpatkové písmo
`\texttt{...}`	`{\ttfamily...}`	`Písmo psacího stroje`
`\textmd{...}`	`{\mdseries...}`	Střední duktus
`\textbf{...}`	`{\bfseries...}`	**Tučné písmo**
`\textup{...}`	`{\upshape...}`	Stojaté písmo
`\textit{...}`	`{\itshape...}`	*Kurzíva*
`\textsl{...}`	`{\slshape...}`	*Skloněné písmo*
`\textsc{...}`	`{\scshape...}`	Kapitálky
`\emph{...}`	`{\em...}`	*Zvýrazněné* písmo
`\textnormal{...}`	`{\normalfont...}`	Základní písmo

Tabulka 2: Definice atributů písma (hooks)

Hook	*Hodnota*	*Použito při:*
`\encodingdefault`	`OT1`	Kódování základního písma
`\familydefault`	`\rmdefault`	Rodina základního písma
`\seriesdefault`	`m`	Duktus základního písma
`\shapedefault`	`n`	Řez základního písma
`\rmdefault`	`cmr`	`\rmfamily` nebo `\textrm`
`\sfdefault`	`cmss`	`\sffamily` nebo `\textsf`
`\ttdefault`	`cmtt`	`\ttfamily` nebo `\texttt`
`\bfdefault`	`bx`	`\bfseries` nebo `\textbf`
`\mddefault`	`m`	`\mdseries` nebo `\textmd`
`\itdefault`	`it`	`\itshape` nebo `\textit`
`\sldefault`	`sl`	`\slshape` nebo `\textsl`
`\scdefault`	`sc`	`\scshape` nebo `\textsc`
`\updefault`	`n`	`\upshape` nebo `\textup`

```
{\bfseries text}
{\fontseries{b}\selectfont text}
{\fontseries{sbc}\sffamily text}
```

Všimněte si, že poslední řádek výše uvedeného příkladu neobsahuje `\selectfont`. Makro `\sffamily` se totiž expanduje na `\fontfamily {\sfdefault}\selectfont`. Poslední řádek je tedy přibližně ekvivalentní zápisu:

```
{\fontseries{sbc}\fontfamily{sf}\selectfont text}
```

Ukázali jsme si, že je naprosto dostačující, když uvedeme `\selectfont` pouze jednou až po definování změn všech požadovaných atributů. Nyní předpokládejme, že chceme jedno slovo napsat v tučné kurzívě rodiny Times, přičemž příslušný font má kódování T1. Měli bychom tedy v dokumentu napsat

```
\fontencoding{T1}\fontfamily{ptm}\fontseries{bx}
\fontshape{it}\selectfont
```

To bychom se ale upsali. Naštěstí nám NFSS nabízí zkrácený zápis:

```
\usefont{T1}{ptm}{bx}{it}
```

Všimněte si, že zde již nepotřebujeme `\selectfont`. Význam parametrů je z ukázky zřejmý.

Nyní již tedy víme, proč funguje `\fontfamily{ptm}\selectfont`. Tento způsob ale není zcela správný. Čísla stránek a běžné záhlaví se tiskne základním písmem dokumentu, takže `\normalfont` přepne zpět na Computer Modern. pokud chceme vysázet celý dokument písmem Times Roman, měli bychom tedy na jeho začátek vložit:

```
\renewcommand\rmdefault{ptm}
```

Tak se chová standardní balík TIMES. V něm jsou navíc změněny rodiny písma bezpatkového a písma psacího stroje.

Nezmínili jsme se ještě o makru `\fontsize`. To slouží ke změně velikosti písma. Makro má dva parametry. Prvním parametrem se definuje stupeň písma, druhý parametr určuje rozteč účaří. V obou parametrech můžeme uvést jak prosté číslo, tak TEXovský rozměr. Pro tisk plakátů nebo folií určených k promítání s oblibou používám:

```
\fontsize{10mm}{13mm}\selectfont
```

Makra `\normalsize`, `\small` a podobná jsou však poněkud složitější. O tom si ale povíme později.

9. Zavedení nového písma

V předchozím textu jsme si vysvětlili, jak máme LaTeXu sdělit, které písmo chceme pro sazbu použít. Mohli jsme ale specifikovat pouze takové písmo, které již LaTeX zná. Nyní si povíme, jak naučíme LaTeX další písma.

9.1. Primitiv `\font`

LaTeX umožňuje použití mnoha TeXových primitivů. Jedním z nich je právě `\font`. Primitiv je velmi rychlý, protože nepodléhá žádným expanzím. Chceme-li tedy z fontu použít jeden nebo několik málo znaků, např. když máme v nějakém fontu vytvořený obrázek, je primitiv `\font` vhodným kandidátem. Pro zavádění písma pro text se však nehodí.

Představme si, že písmo pro azbuku zavedeme pomocí primitivu `\font` takto:

```
\font\azb=wncyr10
```

Pak pomocí `{\azb pisheme bukvy}` skutečně пишеме буквы, protože v daném fontu „sh“ tvoří ligaturu „ш“. Takový font je ale z hlediska LaTeXu statický, takže *v kurzívě* буквы писать нельзя. Stejně tak, маленькие буквы nedokážeme vytisknout.

Ekvivalentním způsobem je i zavedení písma LaTeXovým makrem `\newfont`. Syntaxe je jiná, ale výsledek je stejný.

Podobnou zradu způsobí zavedení fontu s jiným kódováním. Zde jsme přepnuli kódování na T1, takže sázíme fontem `dcr10`. Protože jsme změnili kódování a nemáme česká písmena definována jako aktivní, musíme vše zapisovat TeXovými sekvencemi. Text ale dopadne dobře, neboť se o to LaTeX postará. V následujícím odstavci zavedeme `dcr10` primitivem `\font`. Podívejte se, co z toho vyjde!

Podobnou zradu zpùsobí zavedení fontu s jiným kódováním. Zde jsme pøepnuli kódování na T1, tak¿e sázíme fontem `dcr10`. Proto¿e jsme zmìnili kódování a nemáme èeská písmena definována jako aktivní, musíme vže zapisovat TeXovými sekvencemi. Text ale dopadne dobøe, nebož se o to LaTeX postará. V následujícím odstavci zavedeme `dcr10` primitivem `\font`. Podívejte se, co z toho vyjde!

Kódy některých znaků se shodují, proto je text alespoň částečně srozumitelný. Nicméně, zpravodaj vysázený takovým způsobem by se vám jistě nelíbil. Potřebujeme tedy jiný mechanismus pro zavádění fontů.

9.2. Makro \DeclareFixedFont

Představme si, že nyní zavedeme font dcr10 pomocí:

```
\DeclareFixedFont{\DC}{T1}{\familydefault}{\seriesdefault}
{\shapedefault}{10}
```

Potom jako přepínač písma použijeme \DC a znovu vytiskneme předposlední odstavec předchozí podkapitoly.

Podobnou zradu způsobí zavedení fontu s jiným kódováním. Zde jsme pøepnuli kódování na T1, tak¿e sázíme fontem dcr10. Proto¿e jsme zmìnili kódování a nemáme èeská písmena definována jako aktivní, musíme vše zapisovat TEXovými sekvencemi. Text ale dopadne dobøe, neboť se o to LATEX postará. V následujícím odstavci zavedeme dcr10 primitivem \font. Podívejte se, co z toho vyjde!

Ani tentokrát se to nepovedlo. LATEX sice má informaci o kódování, ale zapomněl jej změnit. Musíme tedy za \DC zapsat ještě \selectfont, čímž se vše opraví. Důkazem je následující odstavec.

Podobnou zradu způsobí zavedení fontu s jiným kódováním. Zde jsme přepnuli kódování na T1, takže sázíme fontem dcr10. Protože jsme změnili kódování a nemáme česká písmena definována jako aktivní, musíme vše zapisovat TEXovými sekvencemi. Text ale dopadne dobře, neboť se o to LATEX postará. V následujícím odstavci zavedeme dcr10 primitivem \font. Podívejte se, co z toho vyjde!

Kódování se nám sice změnilo, ale stejně font zůstává statický. Pro plnohodnotnou definici textového fontu potřebujeme jiný přístup.

9.3. Jak LATEX hledá fonty

Následující text nebude zcela přesný. Pro zajištění efektivity jsou totiž některé fonty interní. Jsou tedy zavedeny již během generování formátu. Málo používané fonty jsou externí. Znamená to, že až při zpracování dokumentu hledá LATEX jejich definici. Pro jednoduchost popíšeme pouze způsob, jak LATEX zavádí externí fonty. S interními fonty se provádí totéž, ale již při vytváření formátu.

LATEX musí rozumět kódování fontu. $\mathcal{CS}$-fonty mají kódování IL2, ale to standardní LATEX nezná. Má-li s takovým fontem pracovat, musí se toto kódování nejprve naučit. LATEX tedy načte soubor IL2enc.def, kde musí být kódování definováno. Obsah tohoto souboru zde popisovat nebudeme. Zájemce odkážeme na standardní dokumentaci, která je povinnou součástí každé distribuce LATEXu.

Předpokládejme nyní, že chceme použít Times-Roman v kódování KOI-8-ČS. Rodinu tohoto písma označujeme zkratkou `ptm`. Někde uvnitř se tedy vyskytnou příkazy `\fontencoding{KOI}` a `\fontfamily{ptm}`. LaTeX se proto porozhlédne po definici kódování KOI-8-ČS. Pokud toto kódování nezná, přečte jej ze souboru `KOIenc.def`. Známe-li definici kódování, můžeme již načítat deklaraci rodiny písma. Ta se v uvedeném případě bude vyskytovat v souboru `KOIptm.fd`. Z tohoto příkladu je snad zřejmé, jak se tvoří názvy fd-souborů pro daná kódování a rodiny fontů. Uživatelé UNIXu si musí uvědomit, že malá/velká písmena v názvech těchto souborů jsou významná. V jiných operačních systémech lze klidně psát i `koiptm.fd`, čímž se ovšem ztrácí přehlednost.

Soubor s deklarací fontu začíná definicí rodiny písma příkazem `\DeclareFontFamily`. Toto makro má tři parametry. Prvním parametrem je kódování, druhým parametrem je název rodiny a na závěr se uvádí příkaz, který se má provést automaticky při zvolení rodiny. Třetí parametr bývá obvykle prázdný. Times-Roman v kódování KOI-8-ČS bychom tedy uvedli definicí:

```
\DeclareFontFamily{KOI}{ptm}{}
```

Poslední parametr lze využít pro změny, které se budou týkat celé rodiny. Například v písmu psacího stroje chceme potlačit dělení. Provedeme to způsobem, který definuje TeX. Při rozdělení slova se za první část vysadí spojovník. TeX ale potřebuje vědět, jak vlastně takový spojovník vypadá. Pro každý font je proto definován parametr `\hyphenchar`, v němž je uložen kód spojovníku. Pokud nastavíme tento kód na -1, znamená to pro TeX, že spojovník neexistuje a dělení se tudíž potlačí. Tuto změnu můžeme provádět automaticky tím, že ji zapíšeme do třetího parametru deklarace rodiny. Při změně rodiny pak LaTeX tuto operaci provede automaticky za nás. Pro písmo psacího stroje tak budeme mít následující definici (v originálním kódování OT1):

```
\DeclareFontFamily{OT1}{cmtt}{\hyphenchar\font=-1}
```

Poslední parametr deklarace rodiny písma lze použít i k jiným trikům. Máme např. písmo, jehož kódování je téměř shodné s IL2, ale některé (nepísmenné) znaky se liší. Mohli bychom definovat nové kódování, ale pak bychom museli znovu vytvořit formát a specifikovat vzory dělení pro další kódování, čímž bychom zbytečně plýtvali pamětí. Protože kódy všech písmen se shodují, můžeme dělit slova podle pravidel určených kódováním IL2. Při zvolení rodiny pouze sdělíme LaTeXu, aby předefinoval

několik symbolů. Pro písmo rodiny `bask` pak taková deklarace vypadá:

```
\DeclareFontFamily{IL2}{bask}{\chardef\textellipsis=8 %
  \def\textsection{^^a7}%
  \def\textdagger{^^83}\def\textdaggerdbl{^^84}}
```

V předchozím textu jsme si popisovali makra pro změnu jednotlivých atributů písma. Uvedli jsme jako příklad `\fontseries{sbc}`, ale nevysvětlili jsme, odkud se to „sbc“ vzalo. Obvykle najdeme v dokumentaci písma, v jakých řezech a duktech je k dispozici. Pokud takovou informaci nemáme, musíme se podívat do fd-souboru a vyčíst ji z příkazů `\DeclareFontShape`. Toto makro má šest parametrů. První čtyři jsou kódování, rodina, duktus a řez. V pátém parametru definujeme velikosti a externí jména fontů. Šestý parametr je obvykle prázdný a specifikují se v něm příkazy, které se mají automaticky provést při zvolení tohoto fontu. Je to obdoba posledního parametru makra `\DeclareFontFamily`.

Nyní se více zastavíme u pátého parametru. Ten totiž poskytuje mnoho různých možností. Předpokládejme, že máme nový font, a chceme LaTeXu sdělit, jak jej má používat. První čtyři parametry zapíšeme snadno, a právě ten pátý nám dá nejvíce práce. Musíme ke každé velikosti definovat jméno fontu, který má TeX zavést. Začneme fiktivním fontem z rodiny `muj` v kódování OT1, který máme k dispozici pouze ve velikosti 10 pt. Pro střední duktus a základní řez pak můžeme psát:

```
\DeclareFontShape{OT1}{muj}{m}{n}{ <10> mujmr10 }{}
```

Máme-li font i ve velikosti 12 pt, použijeme definici:

```
\DeclareFontShape{OT1}{muj}{m}{n}{ <10> <12> mujmr10 }{}
```

Pak při zvolení velikosti 12 pt zavede TeX `mujmr10` ve zvětšení 1,2, což obvykle vyjadřujeme jako `\scaled 1200`.

PostScriptové obrysové fonty jsou libovolně zvětšovatelné a RIP je vygeneruje v libovolné velikosti. Pro všechny velikosti pak máme týž externí soubor a v deklaraci nahradíme konkrétní velikost nekonečným rozpětím. V popisu počeštěného písma Times máme např. pro tučnou kurzívu deklaraci:

```
\DeclareFontShape{IL2}{ptm}{bx}{it}{<-> ptmbi8z}{}
```

Vezměme si nyní příkaz:

```
\DeclareFontShape{IL2}{fnt}{m}{sl}{<-12> fntsmall
                                   <12-> fntlarge}{}
```

Zde jsme definovali skloněné (slanted) písmo fiktivní rodiny `fnt` středního duktu. Pro stupeň menší než 12 pt se zavede externí font `fntsmall`, pro 12 pt a větší se použije `fntlarge`.

V tomto rozsáhlejším úvodu jsme si popsali základní tvar makra `\DeclareFontShape`. Nyní si povíme více o možnostech specifikace velikosti a externího jména. Velikost (stupeň) písma lze zadat buď jednoduchými čísly jako `<10>` a `<14.4>` nebo rozmezím dvou čísel, např. `<5-10>`. Tato specifikace pak platí pro velikost větší nebo rovnou 5 pt a menší než 10 pt. Jednu mez můžeme vynechat. Potom `<-12>` specifikuje všechny velikosti menší než 12 pt, `<14.4->` definuje externí font pro velikost větší nebo rovnou 14.4 pt. Vynecháme-li obě meze, definujeme pak jedním externím fontem všechny velikosti. Jak jsme již ukázali, používá se takový zápis zejména ve spojení s PostScriptovými obrysovými fonty. Font je obvykle dostupný v několika velikostech. Zápis pak můžeme zkrátit vypuštěním opakující se informace. Například písmo Pandora existuje pouze jako jeden METAFONTový soubor, z něhož získáváme další velikosti zvětšováním či zmenšováním. Deklarace pro obvyklé stupně známé z METAFONTu pak vypadá:

```
\DeclareFontShape{OT1}{panr}{m}{n}{
  <5> <6> <7> <8> <9> <10> <10.95> <12>
  <14.4> <17.28> <20.74> <24.88> pan10 }{}
```

Nyní požádáme LaTeX, aby zavedl font pandora ve velikosti 12 pt. Z výše uvedené deklarace NFSS pochopí, že potřebujeme `pan10` ve zvětšení 1,2. Jak ale LaTeX přišel na to, že základní externí font má stupeň 10 pt? Tato informace je uvedena v metrickém souboru `pan10.tfm`. Pokud jej programem TFTOPL převedeme do textové formy, najdeme v `pan10.pl` informaci:

```
(DESIGNSIZE R 10.0)
```

V METAFONTu zadáváme velikost písma pomocí parametrů `font_size` a `designsize`, ale jejich popis patří do jiného článku. Tyto parametry totiž musí znát tvůrce fontu. Chceme-li hotové písmo použít v LaTeXu, nemusíme o jejich existenci vůbec vědět. Je zcela postačující, když si uvědomíme, že velikost písma je zapsána v metrickém souboru a LaTeXové (nebo TeXové – z hlediska uživatele to není podstatné) algoritmy si příslušnou informaci najdou.

Případ, který jsme právě diskutovali, je vlastně nejjednodušší „size function“, tzv. prázdná funkce. Má nejkratší možné jméno, protože se

používá nejčastěji. Existuje řada různých funkcí. Pokud se v zápisu vyskytuje hvězdička, pak vše nalevo od hvězdičky je jméno funkce, napravo jsou argumenty. Některé funkce mají též nepovinný argument, který se pak uvádí v hranatých závorkách. Pokud specifikace velikosti neobsahuje hvězdičku, pak se jedná o prázdnou funkci a celý text je argument.

Základní podobu prázdné funkce jsme si již popsali. Prázdná funkce však může mít jeden nepovinný parametr. Představme si, že máme písmo `mujps`, které chceme kombinovat s fontem Times. Naše písmo má však při stejném nominálním stupni vyšší kresbu a pro estetické sladění bychom jej chtěli redukovat na 93 %. toho dosáhneme automaticky následující definicí velikosti:

```
<-> [.93] mujps
```

Funkce „s“ je funkčně identická s prázdnou funkcí. Jediný rozdíl spočívá v tom, že se žádné diagnostické zprávy nepíší na obrazovku. Do log-souboru se však zapisují vždy. Název funkce je odvozen z prvního písmene anglického slova silence – ticho. Použití funkce „s“ může vypadat například takto:

```
\DeclareFontShape{OT1}{muj}{b}{sl}{ <-> s * [.93] mujps }{}
```

Písma generovaná METAFONTem jsou obvykle dostupná v řadě diskrétních velikostí. Abychom si ušetřili zápis, použijeme s výhodou funkci „gen“ nebo její tichou variantu „sgen“. Zápis:

```
<8> <9> <10> gen * cmtt
```

je pak zkratkou za:

```
<8> cmtt8 <9> cmtt9 <10> cmtt10
```

I tato funkce rozeznává nepovinný argument, který má opět význam zvětšení stejně jako u prázdné funkce.

Některé rodiny písem jsou bohaté na různé řezy a dukty, v jiných rodinách jsou možnosti silně omezeny. Když si v dokumentu zvolíme neexistující kombinaci, NFSS vybere nějaký náhradní font. To ale nemusí být volba, která se nám líbí. Představte si, že sázíte švabachem a požadujete skloněné písmo. Skloněný (slanted) švabach ale nemáte, a NFSS tedy použije skloněný Computer Modern. To je naprosto nežádoucí. Zcela logická je v takovém případě náhrada švabachovou kurzívou, kterou třeba máte k dispozici. Takové substituce lze definovat funkcí „sub“ nebo její tichou variantou „ssub“. Argument této funkce se uvádí ve formě rodina/duktus/řez, kde všechny části označujeme příslušnými zkratkami.

Například v definičním souboru rodiny Times s kódováním IL2 je skloněné písmo nahrazeno kurzívou následujícím příkazem:

```
\DeclareFontShape{IL2}{ptm}{m}{sl}{<-> sub * m/it}{}
```

Písmo Zapf-Chancery-Medium-Italic máme v tiskárně pouze v jediné kombinaci. Téměř celý soubor pak obsahuje substituce:

```
\DeclareFontFamily{IL2}{pzc}{}
\DeclareFontShape{IL2}{pzc}{m}{it}{<-> pzcmi8z}{}
\DeclareFontShape{IL2}{pzc}{m}{sl}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{m}{n}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{m}{sc}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{b}{n}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{b}{sl}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{b}{it}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{b}{sc}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{bx}{n}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{bx}{sl}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{bx}{it}{<-> sub * m/it}{}
\DeclareFontShape{IL2}{pzc}{bx}{sc}{<-> sub * m/it}{}
```

Ve výše uvedených příkladech jsme nahradili celý řez písma řezem jiným. Máme ale další možnost. Předpokládejme, že nějaké písmo je dostupné pouze v některých velikostech. Pokud uživatel zvolí stupeň, který dané písmo nemá, použijeme rozumnou náhradu. Například můžeme mít definici:

```
\DeclareFontShape{OT1}{cmss}{m}{sl}{ <-8> sub * cmss/m/n
  <8> <9> gen * cmssi <10> <10.95> cmssi10
  <12> <14.4> cmssi12
  <17.28> <20.74> <24.88> cmssi17 }{}
```

Fonty velikosti menší než 8 pt budou tedy nahrazeny základním řezem cmss/m/n. Takový řez ovšem neexistuje a definice obsahuje další substituce. Vidíme tedy, že substituce se mohou řetězit, ale pochopitelně nesmějí obsahovat cyklus. Výhoda takového zápisu spočívá v tom, že měníme pouze jednu skupinu, když v nové distribuci získáme nový font.

Další funkce zde již nebudeme popisovat, protože se bez nich obvykle obejdeme. Zájemce odkážeme v závěru článku na příslušnou literaturu.

9.4. NFSS pro starý LaTeX 2.09

Na tomto místě nechceme dělat reklamu starému LaTeXu, ale zastavíme se u něj z jiného důvodu. Přadstavme si, že nám někdo věnuje (nebo prodá) nové písmo a soubor maker pro zavedení tohoto písma právě do NFSS v LaTeXu 2.09. My jsme ovšem pokrokoví a používáme LaTeX 2_ε. Pokud pro nás LaTeX 2.09 není jen obstarožní záhadou, máme většinu práce již hotovu. LaTeX 2.09 neměl fd-soubory, ale nové fonty byly zaváděny pomocí stylů. Makra pro zavedení nového stylu tedy najdete v souboru s příponou `.sty`. Podíváte-li se do takového souboru, bude se vám zdát povědomý. Rodina je tam definována makrem `\@newfontfamily` a jednotlivé řezy jsou specifikovány v `\@newfontshape`. Jejich syntaxe je stejná jako v NFSS2, kterou používá LaTeX 2_ε. Chybí pouze kódování, protože LaTeX 2.09 kódování nerozeznával. Stačí tedy jednoduchá záměna maker a doplnění kódování fontu. Taková operace je v libovolném editoru otázkou několika minut. Výsledný soubor musí mít samozřejmě jméno podle vzoru `KŌDrodina.fd`. Uživatelé OS/2 a UNIXu mohou mít tendenci k extrémně dlouhým jménům souborů. Například si pojmenují rodinu Zapf-Chancery-Medium-Italic plným jménem (bez spojovníků) a pro kód KOI-8-ČS pak dostanou soubor `KOIZapfChanceryMediumItalic.fd`. Vězte však, že takový soubor nebude přenositelný do operačních systémů, kde jsou jména souborů omezena na $8+3$. Pokud jej vůbec nějak dostanete do DOSu, bude se výsledný soubor jmenovat `KOIZAPFC.FD`. V některých implementacích pak taková změna může činit potíže. V nových verzích emTeXu je tento problém (naštěstí) ošetřen.

10. Praktický postup zavádění nového písma

Základní teorii jsme tedy zvládli a můžeme se pustit do práce. Jak ale máme postupovat? Nejlepší je, když nám příslušný soubor napíše nějaký automat. Standardem je balík FONTINST, pro počeštěné fonty lze použít CSPSFONT, jenž je součástí $\mathcal{CS}$TeXu. Může se ovšem stát, že žádný z těchto balíků nemáme, nebo se nám zdá složitý a nechceme se jej učit, případně nemáme všechny informace a potřebujeme narychlo vytvořit alespoň přibližný popis, který půjde v našem dokumentu použít. Čeká nás tedy trocha práce a následující odstavce popíší doporučený postup.

Nejprve musíme identifikovat kódování fontu. Často lze kódování odhadnout z povahy fontu, nebo jej dokonce víme od autora písma. Pokud takovou informaci nemáme, musíme si vytisknout tabulku a srovnat ji s definicemi známých kódů. Dingbaty a jiné exotické fonty obvykle nevyhovují žádnému kódování. Zde musíme použít „neznámé“ (undefined) kódování, které se značí symbolem „U“.

V dalším kroku zvolíme jméno rodiny. Jsou to obvykle první společná písmena jmen souborů. Pokud budeme takové písmo používat výhradně na svém domácím počítači, stačí, když zvolíme pro novou rodinu název, který ještě není použit. Přesto je vhodné držet se zavedených konvencí – viz doporučenou literaturu.

Každé písmo má své metrické soubory s příponou `.tfm`, které musí být součástí distribuce daného fontu. V případě písma vytvořeného METAFONTem se může stát, že dostaneme pouze zdrojové soubory, a metriky si musíme vygenerovat METAFONTem sami. Takový případ je však již řídký. Některá písma mohou obsahovat navíc virtuální skripty. Poznáme to podle toho, že k některým souborům s příponou `.tfm` existuje soubor s příponou `.vf`. Při zkoumání se pak zaměříme právě na tyto soubory. Na metriky, ke kterým nemáme stejnojmenný virtuální skript, zapomeneme. To v žádném případě neznamená, že je můžeme zahodit. Tyto soubory se musí vyskytovat v adresáři, kde je najde TEX a ovladače výstupních zařízení. My se však o ně nebudeme zajímat.

Dešifrováním jmen souborů vyzkoumáme, v jakých řezech a duktech je písmo dostupné. Obvykle ve jménech již najdeme odpovídající písmena `b`, `bx`, `sc` (nebo jen `c`), `ti` (místo očekávaného `it` pro *text italics* – na rozdíl od `mi` pro *math italics*) a konečně `sl` nebo `o` pro skloněné (anglicky *slanted* nebo *oblique*) písmo. Někdy jsou však názvy silně kryptické a nemáme jinou možnost než vytisknout si vzorek písma a z jeho vzhledu usoudit, o jaký řez a duktus se vlastně jedná.

Nakonec potřebujeme zjistit, v jakých velikostech je písmo dostupné. Vezměme si například písmo OCR-B. Podíváme se na metrické soubory a najdeme `ocrb5.tfm`, `ocrb6.tfm`, `ocrb7.tfm`, `ocrb8.tfm`, `ocrb9.tfm` a `ocrb10.tfm`. Číslo obvykle určuje stupeň písma. Protože OCR asi nepoužijeme pro normální text, zvolíme kódování U a rodinu pojmenujeme ocrb. Pak lze psát:

```
\DeclareFontFamily{U}{ocrb}{}
\DeclareFontShape{U}{ocrb}{m}{n}{
  <5> <6> <7> <8> <9> <10> gen * ocrb }{}
```

Pokud by distribuce OCR-B obsahovala pouze metriky a hotové pk-soubory pro výstupní zařízení, byli bychom tím hotovi. My však máme METAFONTový zdroj, takže lze generovat i větší písmo. Pro obvyklé velikosti pak napíšeme:

```
\DeclareFontFamily{U}{ocrb}{}
\DeclareFontShape{U}{ocrb}{m}{n}{
  <5> <6> <7> <8> <9> <10> gen * ocrb
  <10.95> <12> <14.4> <17.28> <20.74> <24.88> ocrb10 }{}
```

Měli bychom ještě nadefinovat substituce pro jiné kombinace duktu a řezu, ale to již necháme čtenářům za domácí úkol. Vše pak zapíšeme do souboru `Uocrb.fd`, který umístíme do adresáře, který prohledává LaTeX. Můžeme však uvedená makra zapsat i do preambule dokumentu. K tomu přistoupíme zejména v případě, že jsme napsali narychlo neúplnou deklaraci. Máme-li `\DeclareFontFamily{U}{ocrb}` v dokumentu, nenačte již LaTeX `Uocrb.fd`, i kdyby tento soubor existoval.

Výše uvedené příkazy jsme zkopírovali těsně nad tento odstavec. Nyní lze zapsat:

```
{\fontsize{14}{18}\usefont{U}{ocrb}{m}{n}%
                                    VZOR OCR 1234567890}
```

Výsledkem je pak:

VZOR OCR 1234567890

Všimněte si, že jsme požadovali stupeň 14 pt, který není k dispozici. LaTeX pak vybere nejbližší existující velikost, v našem případě 14,4 pt. Na to je nutno dávat pozor, neboť někdy to může vést k velmi nepříjemným následkům.

U PostScriptových fontů je situace poněkud jednodušší. RIP jej totiž dokáže vyrastrovat v libovolné velikosti, takže obvykle specifikujeme neomezené rozmezí velikostí `<->`. Článek Pavla Herouta (najdete jej ve Zpravodaji č. 1/96 na straně 43) obsahoval ukázku ve fontu Souvenir. Autor vytvořil svůj článek v LaTeXu 2.09 a poskytl metriky i virtuální skripty příslušného fontu pro kódování XL2. To je vlastně rozšířením IL2, přičemž rozšiřující znaky nebyly nutné. Redakce používá LaTeX 2_ε, a aby mohla být vytištěna ukázka na straně 63, byl během zlomku minuty ručně vytvořen soubor `IL2pso.fd` obsahující:

```
\DeclareFontFamily{IL2}{pso}{}
\DeclareFontShape{IL2}{pso}{m}{n}{<->psol8z}{}
```

```
\DeclareFontShape{IL2}{pso}{m}{it}{<->psoli8z}{}
\DeclareFontShape{IL2}{pso}{m}{sl}{<->psoli8z}{}
\DeclareFontShape{IL2}{pso}{bx}{n}{<->psod8z}{}
\DeclareFontShape{IL2}{pso}{bx}{it}{<->psodi8z}{}
\DeclareFontShape{IL2}{pso}{bx}{sl}{<->psodi8z}{}
```

Pak již fungovaly obvyklé příkazy `\bfseries`, `\itshape` a jiné. Nemáme zde ovšem KAPITÁLKY, protože jejich virtuální skripty nebyly k dispozici. V příkladech použití tohoto fontu jsme je nepotřebovali, proto nebyly v definičním souboru uvedeny jejich substituce. Takový definiční soubor má pak omezenou použitelnost, ale představuje snadné řešení pro případ, že potřebujeme nový font zavést rychle alespoň v některých řezech.

Nyní umíme vytvořit velký nápis fontem Times-Roman stupně 30 pt. Odpovídající font pro obrazovkový prohlížeč ovšem nemáme. Pokud jsme si ale třeba pro Mattesův prohlížeč napsali správně substituční soubor a máme dobře nainstalovaný MFJOB, vhodný náhradní font se automaticky vygeneruje. Na PostScriptové tiskárně pak budeme mít skutečně Times-Roman. Když jej ale zkombinujeme s písmem Computer Modern stejné velikosti, budou nám písmenka poskakovat. Maximální dostupná velikost cm-fontů je totiž 24,88 pt! Lze to napravit zásahem do fd--souboru, ale v matematice to chodit nebude – tam je zavedení nové velikosti podstatně složitější. Mohli bychom i pro matematiku použít Times-Roman, ale ani to není jednoduché. Matematický font musí totiž obsahovat řadu parametrů `\fontdimen`, které program AFM2TFM nevytvoří. Můžeme je sice vytvořit ručně, ale je to pracné a zjištění správných hodnot vyžaduje jistý grafický cit. Potřebujeme-li velké nápisy obsahující matematiku, musíme vše vytisknout v nějaké velikosti podporované LaTeXem a zvštěit celý DVI-soubor parametrem tiskového ovladače nebo PostScriptovou transformací.

11. Změna základní velikosti písma

Všechny standardní třídy dokumentů nastavují stupeň písma na 10 pt. Použitím parametru `11pt` nebo `12pt` lze písmo zvětšit. Co ale můžeme udělat v případě, že požadujeme jinou velikost než 11 pt či 12 pt?

Velikost písma již umíme změnit. Mohli bychom si tedy předefinovat `\normalsize`. Rozhodneme se pro základní stupeň 8 pt s dvoubodovým řádkovým prokladem. Naše definice pak bude:

```
\renewcommand\normalsize{\fontsize{8}{10}\selectfont}
```

Zpočátku bude vše fungovat. Ve složitějším dokumentu však objevíme příliš veliké mezery, například okolo matematických vzorců nebo v enumeracích. LaTeX se totiž snaží „myslet“ za uživatele. Řada rozměrů by měla být v souladu s velikostí písma. Makra pro změnu velikosti fontu na to pamatují. V souboru `size10.clo` je uvedeno:

```
\renewcommand\normalsize{%
   \@setfontsize\normalsize\@xpt\@xiipt
   \abovedisplayskip 10\p@ \@plus2\p@ \@minus5\p@
   \abovedisplayshortskip \z@ \@plus3\p@
   \belowdisplayshortskip 6\p@ \@plus3\p@ \@minus3\p@
   \belowdisplayskip \abovedisplayskip
   \let\@listi\@listI}
\normalsize
\newcommand\small{%
   \@setfontsize\small\@ixpt{11}%
   \abovedisplayskip 8.5\p@ \@plus3\p@ \@minus4\p@
   \abovedisplayshortskip \z@ \@plus2\p@
   \belowdisplayshortskip 4\p@ \@plus2\p@ \@minus2\p@
   \def\@listi{\leftmargin\leftmargini
               \topsep 4\p@ \@plus2\p@ \@minus2\p@
               \parsep 2\p@ \@plus\p@ \@minus\p@
               \itemsep \parsep}%
   \belowdisplayskip \abovedisplayskip
}
```

TeXové primitivy `\abovedisplayskip`, `\abovedisplayshortskip`, jakož i `\belowdisplayskip` a `\belowdisplayshortskip` definují vertikální mezery okolo matematických rovnic. Makra `\topsep` apod. určují rozměry použité ve výčtech. Těm se budeme věnovat v jiném dílu kuchařky, takže je zde ponecháme bez komentáře.

Všimněte si hojného „rojení zavináčů“. Pro větší efektivitu je řada opakovaně se vyskytujících výrazů uložena do maker a registrů. Šetří se tím paměť počítače i čas. Například `\@minus` představuje jediný token, zatímco pro `minus` potřebujeme pět tokenů. Snadno si domyslíme, že `\@xpt = 10pt`, `\@xiipt = 12pt` a `\p@ = 1pt`.

Definici `\normalsize` ale na první pohled nevidíme. Makra pro

změnu stupně písma totiž definujeme prostřednictvím `\@setfontsize`. Tento příkaz má tři parametry: jméno makra, které chceme definovat, stupeň písma a vzdálenost účaří. Přibližně tedy vypadá funkce takto:

```
\def\@setfontsize#1#2#3{%
    \def#1{\fontsize{#2}{#3}\selectfont}}
```

Skutečnost je složitější, ale to nás v tomto okamžiku nebude zajímat. Všimneme si ale toho, že rozměry můžeme zadávat v libovolných jednotkách. Představme si, že chceme vytisknout plakát. Hodilo by se nám základní písmo velikosti 10 mm s dvoumilimetrovým řádkovým prokladem. Vytvoříme si pro tento účel soubor `size10mm.sty`, kde rozměry převedeme z pointů na milimetry. Po triviální úpravě budeme hotovi a soubor bude začínat příkazy:

```
\ProvidesPackage{size10mm}
\renewcommand\normalsize{%
   \@setfontsize\normalsize{10mm}{12mm}
   \abovedisplayskip 10mm \@plus2mm \@minus5mm
   \abovedisplayshortskip \z@ \@plus3mm
   \belowdisplayshortskip 6mm \@plus3mm \@minus3mm
   \belowdisplayskip \abovedisplayskip
   \let\@listi\@listI}
\normalsize
\renewcommand\small{%
   \@setfontsize\small{9mm}{11mm}%
   \abovedisplayskip 8.5mm \@plus3mm \@minus4mm
   \abovedisplayshortskip \z@ \@plus2mm
   \belowdisplayshortskip 4mm \@plus2mm \@minus2mm
   \def\@listi{\leftmargin\leftmargini
               \topsep 4mm \@plus2mm \@minus2mm
               \parsep 2mm \@plus1mm \@minus1mm
               \itemsep \parsep}%
   \belowdisplayskip \abovedisplayskip
}
```

Celý soubor má 217 řádků, proto jej zde neuvádíme. Jistě si zbytek domyslíte sami.

Zdálo by se, že jsme v úvodu této kapitolky zakázali použití makra `\fontsize`. Není to ale pravda. V neobvyklých případech potřebujeme

neobvyklé velikosti písma. Týká se to třeba titulních stránek nebo rozměrných tabulek. Zde bychom si s jednoduchými makry `\normalsize`, `\large`, `\small` atd. nemuseli vystačit. Také můžeme potřebovat jiný řádkový proklad. Tehdy použijeme `\fontsize`, ale nikdy nebudeme takto měnit velikost písma v běžném dokumentu.

Řádkový proklad je častým kamenem úrazu. Budeme se mu věnovat v dalším dílu kuchařky. Tím se vypořádáme s fonty a začneme s popisem dalších triků.

V minulé části jsme slíbili, že ukážeme, jak lze změnit vzhled obsahu. Bohužel se to do tohoto čísla již nevešlo. Zbývá tím dluh, který napravíme v prvním čísle příštího roku.

12. Doporučená literatura

Pro další studium lze doporučit již zmíněný článek Pavla Herouta, kde najdeme skutečné základy. Doporučujeme však, abyste při definicích nepoužívali přímo adresáře. Raději využijte konfigurační soubor, kterým definujete pro TEX i pro ovladače, kde se soubory daného typu vyskytují. Pak bude vaše práce přenositelná i do jiných instalací. Uvítají to vaši kolegové a za odměnu třeba i oni budou vytvářet své produkty s ohledem na přenositelnost a ušetří tím i vaši práci.

Při pokusu o zavedení nového písma se můžete setkat se spoustou problémů ještě dříve, neš se vám podaří vytisknout první písmeno. O těchto problémech a možnostech jejich diagnostiky a odstranění si můžete přečíst v tomto čísle Zpravodaje ve článku Arnošta Štědrého. Článek začíná na straně 249.

Již tradičně doporučíme knihu: Michel Goossens, Frank Mittelbach, Alexander Samarin – *The LaTeX Companion*. Addison Wesley, Reading 1994, ISBN 0-201-54199-8, zejména kapitolu 7.

Užitečné informace jsou i v souboru `usrguide.tex`, který je standardní součástí distribuce LaTeXu.

Zdeněk Wagner
`wagner@mbox.cesnet.cz`

BachoTEX '97

GUST ANNUAL MEETING IN BACHOTEK
"TEX FROM INSIDE"
May 1–3, 1997, Bachotek (Brodnica Lake District, Poland)
PRELIMINARY INFORMATION
OF THE PROGRAMME COMMITTEE

The 5th annual meeting of the Polish TEX Users Group (GUST) will be held in Bachotek, Brodnica Lake District, May 1–3, 1997. We invite TEX users from everywhere to join us. As the theme of the conference suggests, this time we would like to concentrate on the TEX's interior, i. e., on rarely used or even totally unknown aspects of TEX. Submissions on other subjects, such as electronic publishing (PDF, HTML, SGML), typography issues, multilingual issues, graphic, PostScript, etc., are also welcomed. As always, the conference programme will be completed with tutorials covering issues concerning TEX, METAFONT, PostScript, etc.

Paper submissions, proposals for tutorials and remarks related to the programme of the meeting please send to the Programme Committee. Lectures in English are acceptable. The deadline for submissions is 31 March 1997.

For more information about the meeting, please contact GUST secretary Jola Szelatyńska, Ogólnouczelniany Ośrodek Obliczeniowy UMK, ul. Chopina 12/18, 87-100 Toruń, POLAND (email: `mjsz@man.torun.pl`). See also: `http://www.GUST.org.pl`.

Conference Programme Committee:
Tomasz Przechlewski (`ekotp@univ.gda.pl`)
Boguslaw Jackowski (`jacko@ipipan.gda.pl`)
Marek Ryćko (`marek@camk.edu.pl`)

GUST Sectetary
Jolanta Szelatyńska

Přihláška za člena Československého sdružení uživatelů TEXu

Jméno: Příjmení: Titul:

Firma:

Adresa:

Členství:

☐ Individuální ☐ Student

☐ Kolektivní

PSČ:

Telefon: FAX: E-mail:

Podpis:

Přihláška za člena Československého sdružení uživatelů TEXu

Jméno: Příjmení: Titul:

Firma:

Adresa:

Členství:

☐ Individuální ☐ Student

☐ Kolektivní

PSČ:

Telefon: FAX: E-mail:

Podpis:

Přihlášku vložte do ofrankované obálky a odešlete na adresu:

Československé sdružení uživatelů TEXu
c/o FI MU
Botanická 68a
602 00 Brno

FAX: 05–412 125 68

Informace o sdružení lze získat elektronicky prostřednictvím World Wide Webu na `http://www.cstug.cz/cstug`

Přihlášku vložte do ofrankované obálky a odešlete na adresu:

Československé sdružení uživatelů TEXu
c/o FI MU
Botanická 68a
602 00 Brno

FAX: 05–412 125 68

Informace o sdružení lze získat elektronicky prostřednictvím World Wide Webu na `http://www.cstug.cz/cstug`

Vydalo:	Československé sdružení uživatelů TEXu vlastním nákladem jako interní publikaci
Obálka:	Bohumil Bednář
Počet výtisků:	750
Uzávěrka:	31. října 1996
Odpovědný redaktor:	Zdeněk Wagner
Tisk a distribuce:	KOPP, Šumavská 3, 370 01 České Budějovice
Adresa:	CSTUG, c/o FI MU, Botanická 68a, 602 00 Brno
fax:	05–412 125 68
e-mail:	cstug@cstug.cz

Zřízené poštovní aliasy sdružení CSTUG:

`bulletin@cstug.cz`, `zpravodaj@cstug.cz`
  korespondence ohledně Zpravodaje sdružení

`board@cstug.cz`
  korespondence členům výboru

`cstug@cstug.cz`, `president@cstug.cz`
  korespondence předsedovi sdružení

`cstug-members@cstug.cz`
  korespondence členům sdružení

`cstug-faq@cstug.cz`
  řešené otázky s odpověďmi navrhované k zařazení do dokumentu CSFAQ

`secretary@cstug.cz`, `orders@cstug.cz`
  korespondence administrativní síle sdružení, objednávky CD-ROM

`bookorders@cstug.cz`
  objednávky tištěné TEXové literatury na dobírku

ftp server sdružení:
  `ftp://ftp.cstug.cz/`

www server sdružení:
  `http://www.cstug.cz/`

Podávání novinových zásilek povoleno Oblastní správou pošt
v Českých Budějovicích, j. zn. P 3.202/94 ze dne 19. července 1994