# Tvorba písma pomocí programu METAFONT

# Creation of the fonts using METAFONT

**Bakalářská práce**

**Autor: Václav Falc**

**Vedoucí „závěrečné“ práce: Mgr. Jiří Pech Ph. D.**

**Jihočeská univerzita v Českých Budějovicích**

**Pedagogická fakulta**

**Katedra Informatiky**

**2008**

## Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě pedagogickou fakultou elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách.

V Českých Budějovicích dne 25. 4. 2008

## Anotace

Práce popisuje program METAFONT. Obsahuje několik základních informací o programu TEX. Popisuje, jak v programu METAFONT vytvořit znakovou sadu. Jako příklad znakové sady jsem vytvořil runové písmo (starší FUTHARK).



## Abstract

This work describes program METAFONT. Work contains some basic informations about program TEX. It describes, how create in program METAFONT font. I created rune letters (older FUTHARK) as example of font.

# Poděkování

Chtěl bych poděkovat vedoucímu práce Mgr. Jiřímu Pechovi PhD, za pomoc a poskytnuté rady při tvorbě mé práce a za poskytnutí implementace TEXu.

# Obsah

# 1 Úvod

Na úvod této práce bych chtěl uvést, komu je kniha určena a co v ní lze nalézt. Tato kniha se může hodit každému, kdo má zájem o práci s programem METAFONT, ale jeho znalost anglického jazyka je malá. Když jsem hledal na internetu podklady pro tuto práci, nenašel jsem žádný dokument popisující v češtině, jak pomocí programu METAFONT vytvořit znakovou sadu a jak ji poté zavést do programu TEX. Existuje pouze několik manuálů, které popisují kreslení tímto programem, třeba [5]. V knize je i velmi zevrubně popsán program TEX, ale pokud se o něm chcete dozvědět více, nahlédněte do použité literatury, kde je několik textů, ze kterých jsem čerpal i já a ve kterých naleznete více informací o této problematice.

Hlavním úkolem při tvorbě této práce bylo vytvoření znakové sady pomocí programu METAFONT a použití této znakové sady v programu TEX. Prvním problémem, na který jsem narazil, bylo jakou znakovou sadu vytvořit? Po několikadenním přemýšlení a zavrhnutí několika nápadů jsem se rozhodl pro vytvoření znakové sady runových písmen (starší FUTHARK). Důvodů bylo hned několik: samotnému se mi toto písmo líbí a zajímá mě i severská mytologie.

Výsledky mé práce může použít každý, kdo bude chtít svůj text napsaný programem TEX obohatit o runy, nebo pro každého, kdo runy na něco potřebuje využít. Případně pro každého, kdo se chce seznámit s programem METAFONT.

Práce je rozdělena na tři hlavní části:

1. část (druhá kapitola) - TEX. V ní je jen zevrubný popis několika částí problematiky týkající se TEXu. Všechny části uvedené v této kapitole jsou zde pouze proto, aby byla alespoň malá představa o tom, jak program TEX pracuje, jak používá fonty, a aby bylo možné vytvořenou znakovou sadu v tomto programu použít.

2. část (třetí kapitola) - METAFONT popisuje podrobněji práci s programem METAFONT a je v ní i několik ukázek kódu.

3. část (čtvrtá kapitola) - popisuje, jak jsem tvořil runové písmo pomocí programu METAFONT a jeho zavedení do programu TEX. Na závěr této kapitoly jsem uvedl ještě několik ukázek kódů v programu METAFONT, protože při tvorbě runového písma se využije jen malá část schopností tohoto programu.

# 2 TEX

## 2.1 Co je to TEX?

TEX je typografický systém napsaný Donaldem E. Knuthem ze Stanfordské univerzity. Je určen pro velmi kvalitní sazbu knih s množstvím matematiky. Svůj produkt D. E. Knuth daroval světu zdarma roku 1983, kdy byla dokončena verze 3. Od té doby nebyla v systému provedena žádná zásadní změna jeho koncepce.

Název TEX se vyslovuje [tech], protože anglické slovo *technology* pochází z řeckého kořene začínající písmeny *Tex*; toto slovo v řečtině znamená nejen technologii, ale i umění. Program lze používat i pro nelatinkové abecedy, lze v něm sázet zprava doleva, nebo i vertikálně. TEX je implementován na mnoha platformách (DOS, Linux, Windows, atd.). Většina těchto implementací je freeware, ale existují i komerční produkty.

Naučit se používat TEX je tak trochu jako naučit se cizí jazyk. Nejprve se zdá, že se člověk musí naučit úplně ohromující řadu nových termínů, z nichž některé patří do typografie a jiné připomínají věci z programování. Nejúplnějším zdrojem informací o TEXu je nepochybně knížka The TEXbook od Donalda E. Knutha. Tato kniha je úplnou encyklopedií a slovníkem pro uživatele TEXu a měl by ji mít k dispozici každý, kdo TEX pravidelně používá.

Uživatelé TEXu mají svá sdružení (většinou podle národnosti); prvním byl americký TUG (TEX User’s Group), u nás vznikl CSTUG (Czech and Slovak TEX User’s Group). Tato organizace vytváří a šíří instalační balík TEXu, který je přizpůsoben české a slovenské sazbě.

## 2.2 Jak TEX pracuje?

TEX je formátor. Je to program, kterému předložíme vstupní text dokumentu v „holé“ textové podobě (bez skrytých formátovacích informací) doplněný textovými značkami, které vymezují strukturu dokumentu nebo dávají TEXu pokyny o způsobu formátování dokumentu. Tento soubor se obvykle nazývá vstupní soubor (TEX file). Bývá obvyklé (ale není to nutné) pojmenovat tento soubor s použitím přípony .tex, například dokument.tex.

Na výstupu pak po zpracování TEXem dostaneme soubor s extenzí tzv. dvi soubor, který obsahuje definitivní informace o použitých fontech a o umístění jednotlivých znaků na stránce. Koncovka souboru .dvi má svůj význam. Jedná se o zkratku, která je odvozena od DeVice Independent (nezávislý na zařízení). Soubor dvi má název odvozený od názvu vstupního souboru, ale s příponu .dvi. Lze jej prostřednictvím vhodného programu, který nazýváme ovladač (driver, resp. device driver), prohlédnout na obrazovce (vidíme, jak bude dokument skutečně vypadat), nebo jej vytisknout na tiskárně. Jestliže jste vytvořili soubor dvi, který dává správný výstup například na obrazovce, můžete si být jisti, že dá přesně totéž (samozřejmě v kvalitnější podobě) na laserové tiskárně. To při přípravě textů umožňuje odstranit velkou část chyb před prvním vytištěním. Celý proces lze chápat jako postup podle schématu:


obrázek 1

Schéma práce TEXu

Existují dva způsoby zpracování vstupního souboru TEXem: dávkový (tzv. batch mode) nebo interaktivní. Při dávkovém způsobu zpracování

předložíte soubor počítači ke zpracování a na konci zpracovávání dostanete výsledek. V interaktivním způsobu se běh programu může přerušit pro přijetí dalších instrukcí od uživatele, tj. uživatel může být v interakci s programem. Interaktivní použití TEXu umožňuje uživateli opravu některých chyb, v dávkovém módu TEX opravuje chyby sám, jak nejlépe umí. Prvému způsobu se dává samozřejmě přednost. Všechny implementace TEXu na osobních počítačích a většina implementací na velkých počítačích umožňují interaktivní zpracovávání, na některých velkých počítačích je ale možné jen zpracování dávkové.

TEX tedy čte na svém vstupu soubor s dobře definovanou syntaxí jazyka značek a na výstupu je soubor s úplným popisem sazby. TEX jako takový je nezávislý na operačním systému. Vývoj samotného TEXu je zastaven, takže pro uživatele nehrozí nebezpečí vzniku dalších nekompatibilních verzí.

## 2.3 Implementace TEXu

Další programy „okolo TEXu“ tvoří dohromady tzv. implementaci TEXu a umožní s TEXem rozumným způsobem pracovat.

Začínající uživatel se samozřejmě hlavně ptá po způsobu, jak může v konkrétním operačním systému s konkrétní TEXovou implementací pracovat. Ptá se tedy po uživatelském rozhraní. Jednotlivé manuály o TEXu tradičně odkazují na tzv. „místní příručku“ (Local Guide), která by měla toto rozhraní popisovat. Tato příručka je závislá na použitém operačním systému, na použité implementaci TEXu, na vybraném textovém editoru a někdy též na administrátorovi systému, který konfiguruje některé věci pro větší pohodlí uživatelů.

V graficky orientovaných operačních systémech je většinou možné otevřít současně v jednom okénku textový editor, ve kterém uživatel píše nebo modifikuje vstupní text a ve vedlejším okénku prohlížeč výstupního dvi souboru. Po modifikaci vstupního textu a spuštění TEXu se během pár sekund projeví změna v prohlížeči výsledného dokumentu.

Textový editor, ve kterém připravujeme nebo modifikujeme vstupní texty dokumentů, nesmí ukládat na disk žádné skryté formátovací informace implementované jen pro tento editor (jako například změna fontu, měkké konce řádku apod.). To dělají tzv. textové procesory, které v případě práce s TEXem většinou nepoužíváme.

Zvyklosti ve značkování dokumentu jsou vesměs závislé na použitém formátu TEXu, který modifikuje jeho chování. Říkáme, že je dokument napsán ve formátu LaTEX, pokud je někde na začátku vstupního textu dokumentu uvedena značka *documentclass* nebo *documentstyle*. Pokud tam tuto značku nenajdeme, můžeme předpokládat, že je dokument napsán ve formátu plain, v případě česky psaného dokumentu pak ve formátu csplain. Následující obrázek ukazuje způsoby spuštění TEXu. Předpokládáme, že je k dispozici operační systém, který umožňuje uživateli zadávat pokyny z příkazového řádku. Uvažujme, že je vstupní text dokumentu připraven v souboru dokument.tex.

| příkazový řádek | komentář |
|---|---|
| `tex dokument` | anglický dokument, formát plain |
| `csplain dokument` | český nebo slovenský dokument, formát csplain |
| `latex dokument` | formát LaTeX, jazyk je deklarován v dokumentu |
| `cslatex dokument` | modifikovaný LaTeX pro češtinu a slovenštinu |

obrázek 2

Způsoby spouštění TEXu

Všimněte si, že v příkazovém řádku píšeme za jméno formátu název vstupního souboru a že příponu .tex nemusíme psát. Dobře nainstalovaná implementace TEXu by měla podle jména formátu spustit TEX modifikovaný právě tímto formátem. Pokud nemáme ve své implementaci TEXu formát csplain, je to špatné. Nebudete schopni překládat texty obsahující české znaky.

K prohlédnutí dokumentu se používají různé programy (tzv. dvi prohlížeče). Volba prohlížeče závisí na použitém operačním systému a na implementaci TEXu. V UNIXu je nejčastější program xdvi (stačí napsat xdvi dokument). Ve Windows se používá windvi, dviwin nebo yapp.

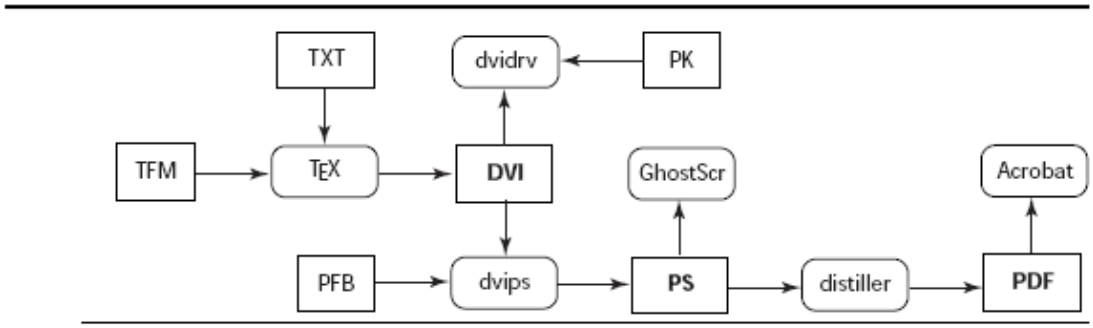
## 2.4 Práce TEXu s fonty

Jádro systému tvoří překladač jazyka TEX, jeho vstupem je textový soubor pořízený libovolným editorem. Do tohoto souboru se vedle vlastního textu zapisují také příkazy, které určují, jak má být text vysázen. Hlavní prací překladače je rozmístění jednotlivých znaků do sazebního zrcadla. K tomu potřebuje znát rozměry znaků. Všechny znaky jsou v tomto okamžiku chápány jako obdélníky, jejichž rozměry jsou soustředěny v souborech s koncovkou .tfm (TEX Font Metric). Průběh překladu a všechna varovná a chybová hlášení jsou překladačem vypisována na obrazovku a současně i do textového souboru s koncovkou .log.

Kódování v TEXu můžeme rozdělit do dvou oblastí: kódování dokumentu a kódování fontů. TEX pracuje ve svém vnitřním kódování, do kterého jsou převáděny znaky z kódování dokumentu pomocí dvou (vstupní a výstupní) vektorů. Vnitřní kódování TEXu je nezávislé na operačním systému. Nastavení kódovacích vektorů je často možné jen ze zdrojových souborů před kompilací TEXu. Některé implementace umožňují tyto vektory měnit i po kompilaci. V EmTEXu tcp tabulkami v podobě inicializačního

formátu nebo na UNIXových instalacích prostřednictvím EncTEXu až za běhu TEXu.

Jak bylo uvedeno výše, TEX nepracuje s vlastními obrazy písmen (fontů), ale pouze s jejich metrikami. Tyto metriky jsou uloženy v souborech *.tfm (TEX fonts metrics). Pokud chceme, aby TEX sázel v nějakém nestandardním fontu, který nebývá součástí TEXovských instalací, pak principiálně stačí získat z tohoto fontu metrické informace a transformovat je do formátu TFM. Pak musíme použít dvi-ovladač, který s těmito fonty umí pracovat.




obrázek 3

Práce TEXu, ovladačů a převodníků

Kódování fontů je stejné jako kódování dvi souboru, které lze modifikovat pomocí virtuálních fontů. Virtuální fonty obsahují návod, jak namapovat znaky ze souboru s jiným kódováním do našeho kódování nebo jak sestavit znak na dané pozici ze znaků jiných fontů. Možnosti virtuálních souborů jsou využívány při používání postscriptových a TrueType fontů v TEXovské sazbě. [3]

## 2.5 Základní příkazy pro práci s fonty

Při zavedení TEXu je implicitní font Computer Modern, antikva ve velikosti 10pt. Plain TEX poskytuje následující příkazy pro změnu řezu:

| | | |
|---|---|---|
| \rm | normální písmo | Roman |
| \sl | skloněné písmo | *Slanted* |
| \it | kurzíva | *Italic* |
| \tt | psací stroj | `Typewriter` |
| \bf | tučné písmo | **Bold face** |

obrázek 4

Příkazy pro změnu řezu

Zavedení nového fontu v plain TEXu:

`\font\jméno=`externí jméno fontu `at` požadovaná velikost

příklad:

```
\font\malyroman=csr10
```

Příkaz *\font* zavede do paměti font ze souboru *.tfm (např. csr10.tfm) a přiřadí mu jméno (malyroman), které bude použito jako přepínač pro aktivaci fontu. Chceme-li zvětšený font, použijeme příkaz *at* a za ním požadovanou velikost. Násobnou změnu velikosti lze provést příkazem *scaled*. *scaled 1000* znamená 1:1 (tedy žádné zvětšení), *scaled 2000* provede dvojnásobné zvětšení, *scaled 750* dodá písmena o 75 % původní velikosti.

V typografii je časté odstupňování písma koeficientem 1,2. Příkaz *\magstep2* vyjadřuje základní velikost ×1,44. Změna velikosti fontu znamená pro TEX pouze pronásobení metriky. Velikosti písma se udávají číslem a označením pro měrný systém. V našem geografickém prostoru je domovský didotův měrný systém.

obrázek 5

Zkratky měrných systémů

| | |
|---|---|
| pt | point |
| pc | pica (1 pc = 12 pt) |
| in | inch (1 inch = 72.27 pt) |
| bp | big point (72 bp = 1 inch) |
| mm | milimetr (10 mm = 1 cm) |
| cm | centimetr (2,54 cm = 1 inch) |
| dd | didotův bod (1157 dd = 1238 pt) |
| cc | cicero (1 cc = 12 dd) |
| sp | scaled point (65536 sp = 1 pt) |

# 3 METAFONT

## 3.1 Co je to METAFONT?

Možná nejlepším způsobem jakým pochopit, co to METAFONT vlastně je, bude pochopení jeho názvu.

Z výkladem názvu začneme jeho koncem FONT. Významu slova font je několik, nás ale zajímá význam z pohledu tvorby fontů. Jednoduše by se dalo říct, že font je kolekce znaků podobného „stylu“, nebo podobného typu písma v typografické řeči. Font je specifická realizace jistého typu písma podle nějakých společných parametrů, které můžou být třeba výška, šířka (normální, tlusté), styl (normální, kurzíva, písmo se stejnou roztečí), přítomnost patek (malé dekorativní čárky na koncích písmen).

Nyní k části META-. Meta je předpona z řečtiny, která původně znamenala „poté“. Význam slova se časem změnil na význam „vyšší řád než“. V latině a v moderních jazycích (kromě řečtiny, v ní platí stále původní význam slova) máme třeba výrazy metajazyk (jazyk pro popis jazyků), metahistorie (studium o tom, jak lidé viděli historii a jak ji studovali), metapravidlo (pravidlo o pravidlu). Je ještě jedno další místo, kde je výraz meta skutečně platným, a to je tvorba fontů. Konvenční font design znamená, že každou hodnotu parametru musím nejdříve určit, ale na druhou stranu všechny parametry si můžu vymyslet sám. Tyto parametry musím mít pro každý znak znakové sady (to jest několik desítek, nebo stovek znaků), a vždy pro každý řez písma (aby fonty mohly být použity v jakékoliv velikosti). Dokonce za pomoci moderních nástrojů jako jsou programy pro tvorbu znaků je to nudná práce. Místo toho by bylo skvělé mít meta-design. Design, který je nezávislý na hodnotách parametrů. Místo návrhu fontu pro každou kombinaci hodnot parametrů, byste popsali obraz písmene jednou. Seznamem

jednotlivých vstupních parametrů a nechali počítač automaticky dotvořit všechny fonty, jaké potřebujete, se všemi kombinacemi parametrů a když zapomenete jednu kombinaci, tak místo rekonstrukce celé znakové sady jen změníte nějaké parametry a necháte počítač udělat nudnou práci znovu za vás.Touto cestou se ušetři hodně práce a je zajištěn jednotný vzhled každého z vašich písmen.

Jak už asi tušíte METAFONT je přesně o meta-designu. METAFONT je systém, který vám umožní popsat font jednou, a vytvořit tak další fonty z tohoto obrazu písma jen výměnou sady dobře vybraných parametrů, které jsou odděleny.

## 3.2 Jak METAFONT pracuje?

METAFONT je systém, pomocí něhož uděláte stovky podobných fontů jednoduše a s minimem vaší tvůrčí práce. Jak tedy METAFONT funguje? Možná to překvapí, ale METAFONT je vlastně jednoduchá aplikace pro příkazovou řádku, která se na většině platformách volá *mf*. Program nemá žádné grafické rozhraní. Tak jak vlastně udělat nějaké písmeno, když se v METAFONTU nemůže nakreslit? Jednoduše: program je jen interpret. Vezme si sérii instrukcí jako vstup a ty jednoduše zpracuje. Jinými slovy METAFONT je programovací jazyk jako třeba C, BASIC, Pascal (v Pascalu byl napsán kód programu METAFONT) a samozřejmě TEX.

Jak udělat váš popis znaku, aby ho byl mf schopný interpretovat? Možných cest je několik, ale nejběžnější cestou je vytvořit soubor, který obsahuje vaše instrukce a přidat souboru koncovku mf. Soubor musí být pouze čistý text. METAFONT jej přečte a pokud je vše v pořádku vrátí dva odlišné soubory se stejným názvem, ale každý s jinou koncovkou, tomuto procesu se říká kompilace. Jeden soubor bude mít koncovku .tfm (font metrics file). Tento soubor obsahuje informace o velikosti znaků. Druhý soubor bude mít

koncovku ve formě .<číslo>gf, kde <číslo> je určitá hodnota obvykle o třech, nebo čtyřech číslicích. V tomto soubory jsou skutečné tvary písmen.
K funkčnosti fontu jsou oba tyto soubory nezbytné.

Bylo by krásné, kdyby to bylo až takhle snadné. Z historických důvodů, soubor gf není nakonec vlastně využit. Je třeba nejdříve první soubor přetvořit do menšího souboru, obsahující stejnou informaci, ale v zabalené formě. To uděláte pomocí druhého programu. Ten se jmenuje GFtoPK. Výsledný soubor má stejné jméno jako původní soubor, až na to, že jeho přípona nyní vypadá .<číslo>pk (se stejným číslem jako gf soubor). Soubory pk a tfm tvoří vlastní font. Poslední věc, kterou musíte dělat je dát je na místo, kde je rozpozná TEX a nebo LATEXem. To uděláte jednou a můžete vesele používat vaše fonty v TEXu a nebo LATEXu pořád.

Zdá se vám to komplikované? Je to stejné jako z mnoha jinými věcmi. Ve skutečnosti je všechno ještě o něco komplikovanější, než jak jsem to vysvětlil. Nakonec ještě uvedu shrnutí postupu:

1) Napsat soubor obsahující METAFONTové instrukce a uložit ho s koncovkou .mf.
2) Zavolat mf na tento soubor. Výsledkem budou dva soubory. Jeden koncovkou .tfm a druhý s koncovkou .<číslo>gf.
3) Zabalit soubor gf pomocí GFtoPK. Tím dostaneme soubor s koncovkou .<číslo>pk.
4) Přesunout soubory tfm a pk na místo, kde je rozpozná TEX, případně LATEX.
5) Užívat si vlastní písmo ve svém dokumentu.

## 3.3 Něco o algebře

Asi by se chtělo zmínit, proč je zde právě tato kapitola. Pokud chceme program METAFONT využívat v jeho plném potenciálu, tak se bez algebry neobejdeme. Co si pod pojmem algebra představit? Zjednodušeně algebra je představa o souvisejících neznámých, které můžeme označit a psát mezi těmito

neznámými vztahy, na které se poté můžeme odkazovat jako na proměnné a formovat mezi nimi rovnosti. Tímto způsobem přesně METAFONT pracuje. Vy napíšete rovnosti a program si je sám přebere a vyřeší. Abychom si mohli vyzkoušet práci METAFONTu s rovnicemi spustíme jej přes příkazovou řádku spustit příkazem *mf*. Vypíše se nám něco jako:

```
This is METAFONT, Version 2.71828 (MiKTeX 2.3)
**
```

„**“ značí, že METAFONT očekává jméno nějakého souboru, který by přeložil. To však nyní nepotřebujeme. Proto napíšeme příkaz *\relax*. Nyní se nám „**“ změní na „*“. METAFONT nyní bude pracovat s instrukcemi zadanými z příkazové řádky.

Pro lepší pochopení práce METAFONTu s rovnicemi ukážeme si něco na příkladu. Nyní do příkazové řádky napište:

```
tracingequations:=tracingonline:=1;
```

Tímto METAFONT požádáte, aby vám ukázal jak pracuje na pozadí, když mu dáte rovnici k vyřešení. Nyní napište:

```
a+b-c=0;
```

METAFONT odpoví:

```
## c=b+a;
*
```

METAFONT poznal, že se jedná o vztah mezi nějakými neznámými a přeskupí si je jak uzná za vhodné. Poté napište:

```
c=2a;
```

METAFONT ví o matematické notaci dost na to aby poznal, že zápisem „*2a*“ myslíte „*2*a*“. Na rozdíl od mnoha programovacích jazyků METAFONT chápe tuto zkratku a odpoví:

```
## b=a;
```

Při procesu řešení daných rovností nahradí nejprve v první rovnici hodnotu „*c*“, tím, že jsme mu dali druhou rovnost vytvoří jednodušší rovnost závisející jen na „*a*“ a „*b*“. Nakonec napište:

```
a=5;
```

METAFONT odpoví:

```
## a=5
#### b=5
#### c=10
```

METAFONT tím říká, že po zadání aktuální hodnoty „a“ je schopen díky zadaným rovnostem dopočítat hodnoty „*b*“ a „*c*“, nyní jsou všechny proměnné známé. Vy jste zadali systém rovností a METAFONT ho vyřešil a vrátil výsledky. Díky tomu, že METAFONT udělá toto vše sám, není třeba až tak moc algebry.
Nyní ale napište:

```
c=0;
```

METAFONT se to nezalíbí a odpoví:

```
! inconsistent equation (off by 10).
<to be read again>
;
```

```
<*> c=0;
```

Co to znamená? Stejně jako u jiných programovacích jazyků „=“ není operátor přiřazení, ale vztahový operátor. To znamená, že když napíšete „*c=0;*“ neříkáte tím „dej c hodnotu 0“, ale „dávám ti rovnici, kterou tvoří vztah mezi proměnnou c a hodnotou 0“. Projitím ostatních rovností najde, že proměnná „*c*“ má již specifickou hodnotu (v našich ukázkách 10) a nová rovnost popírá předchozí výsledky a je v rozporu i se zbytkem, a to způsobí chybu. Tato vlastnost společně se schopností řešit rovnice v pořadí, ve kterém byly zadány, je hlavní rozdíl mezi METAFONTem a většinou ostatních programovacích jazyků. Většina programovacích jazyků je imperativní, to znamená, že vy zadáte pořadí, v jakém dosáhnout nějakého cíle skrze příkazy (v těchto jazycích „*c=0*“ znamená vyřaď předchozí hodnotu „*c*“ a přiřaď jí novou hodnotu). METAFONT je ale jazyk deklarativní, vy neurčujete, jak najít výsledek, ale popíšete jen věci potřebné k nalezení výsledku a program si najde cestu sám přes zadané vztahy. Takto jste nechali špinavou práci na METAFONTu a nemusíte se zabývat detaily. Tím dosáhneme plné výkonnosti METAFONTu bez dělání zbytečné práce, kterou by za nás dělal stejně sám a rychleji.

## 3.4 Souřadnice

Doposud padla řeč jen o proměnných, rovnicích a číslech, ale METAFONT je o kreslení písmen. Teď tedy něco o tom, jak se to dělá.

Jak METAFONT ví, kde se má co nakreslit? K tomu aby věděl, kde má co kreslit využívá souřadnice. Každý určitě něco o souřadnicích už slyšel. Třeba pozice na mapě se dá určit pomocí dvou souřadnic (zeměpisná šířka a zeměpisná délka). Stejný princip jako v zeměpise mají souřadnice

i v METAFONTu. Příklad: když si vezmete čtverečkovaný papír a určíte si na něm referenční bod, poté co je zvolen, jste schopni jakékoliv místo na papíře lokalizovat a to dvěma čísly - vzdáleností doprava (x-ová souřadnice) a vzdáleností nahoru (y-ová souřadnice). Pokud je bod nalevo, nebo pod referenčním bodem, použijí se záporná znaménka. Takže bod se souřadnicemi (5, -10) bude vzdálen 5 jednotek vzdálenosti od referenčního bodu a 10 jednotek vzdálenosti pod referenčním bodem.

METAFONT používá svůj vlastní „čtverečkovaný papír“ skládající se z mřížky čtverců (pixelů). Jako základní jednotka pro souřadnice se používá pixel. Pokud jde o pozici referenčního je určena absolutně.

## 3.5 Rovné čáry

Čára je kolekcí bodů. Čára (křivka) jakéhokoliv tvaru nebo délky obsahuje nekonečný počet bodů, a proto je mnohem těžší popsat ji než bod, na jehož popis nám stačí dvě souřadnice. Naštěstí je METAFONT dost inteligentní na to, aby většinu práce udělal za nás. Instrukcí pro vykreslení čáry (křivky) v programu METAFONT je instrukce „draw”. I když to tak nevypadá, je to velmi silný nástroj. Stačí zadat kolekci bodů, přes které má čára (křivka) vést a křivka se vykreslí přes body v pořadí, v jakém byly zadány. Například pokud chceme vykreslit vodorovnou čáru dlouhou 100 pixelů a vycházející z referenčního bodu stačí do konzole napsat:

```
draw(0,0)..(100,0); showit; sholit; end
```

Pro vykreslení čáry (křivky) jsou potřeba minimálně dva body. První bod, ze kterého vychází a druhý, ve kterém bude končit. O zbytek se postará program sám.

## 3.6 Středník

Jak jste si již asi všimli, všechny příkazy jsou v METAFONTU ukončeny středníkem. Středník zde má stejný význam, jako v jiných programovacích jazycích (C, Java, Pascal), ukončení příkazu. Pokud příkaz nebude ukončen středníkem, nebude se to METAFONTu „líbit“.

## 3.7 Souřadnice jako proměnné

Jde se nějak odvolávat na bod s neznámými souřadnicemi? Můžeme vzít dvě neznámé proměnné „*a*“ a „*b*“ a požít je jako souřadnice k definici bodu o souřadnicích (a,b). Pokud je ale zapotřebí více bodů, brzy ztratíme orientaci, která souřadnice je jakého bodu. Naštěstí i v tomto případě nám METAFONT pomůže. Pokud přidáme k názvu bodu číslici, třeba 1, jeho x-ová souřadnice bude automaticky $x_1$ a jeho y-ová souřadnice bude $y_1$. Výhodou je, že nemusíte deklarovat jakékoliv ukazovátko označené 1. Jakmile se budete odvolávat k souřadnicím $x_1$, nebo $y_1$, METAFONT pozná, že se označuje k bodu označeném 1. Díky tomu můžete definovat body za použití jejich souřadnic a budete vědět, co k čemu patří. Pár příkladů:

```
(x1, y1) = (0, 0);
(x2, y2) = (5, 0);
(x3, y3) = (0, 5);
```

Jak je na příkladech vidět, je možné definovat přímo dvojice souřadnic za požití „=“ instrukce.

Je sice výhodné využít tohoto zápisu, ale jak to bude výhodné, pokud budete potřebovat zapsat třeba třicet bodů? Asi moc ne, třicetkrát psát ($x_i$, $y_i$) by bylo docela otravné. V METAFONTu to lze ale i jinak, a to pomocí výhodných

zkratek. Zápis z1 je naprosto shodný se zápisem ($x_1$, $y_1$). Takže předchozí ukázka zapsaná pomocí zkratek:

```
z1 = (0, 0);
z2 = (5, 0);
z3 = (0, 5);
```

Tím jsme zápis trochu zkrátili, ale ne o moc. O skutečné výhodě tohoto zápisu je následující kapitola.

## 3.8 Algebra s pomocí souřadnic

Nyní, když máme definované proměnné pro x-ové a y-ové souřadnice bodů, nic nám nebrání v tom je definovat ve stejnou dobu. Dokud nebudete bod využívat při nějakém vykreslování, METAFONTu bude jedno, že x-ová souřadnice je definovaná o 100 řádek v kódu výše než y-ová. Souřadnice jsou proměnné v jejich pravém slova smyslu, mohou být použity v rovnicích a můžou být definovány řešením těchto rovnic.
Příklad:


obrázek 6
Obdélník

Obrázek se skládá ze dvou párů stran, jeden pár tvoří horizontální strany a druhý pár vertikální strany. Délka stran nás teď nezajímá, jen nám jde o to, aby vznikl obdélník. Délky stran můžou záviset na nějaké rovnici, kterou si METAFONT může vyřešit sám. V jiných programovacích jazycích musíte vědět, že chcete obdélník a bezchybně nadefinovat, kde mají být rohy tohoto obdélníku. Ale METAFONT není nějaký z jiných programovacích jazyků a nemusíte si dělat práci s tím, aby vám vznikl obdélník, udělá to za vás. Jak? Podívejte se na obrázek obdélníku, nenapadají vás nějaké obecné rysy, které nemají nic společného s délkou jeho stran? Co třeba pozice rohů obdélníku?

V obdélníku jsou body *1* a *2* ve stejné horizontální hladině, stejně jako body *3* a *4*. Body *1* a *4* jsou na stejné vertikále stejně jako body *2* a *3*. Takže nám stačí akorát si pamatovat, že horizontální zarovnání bodů mají stejnou y-ovou souřadnici, zatímco vertikální zarovnání mají stejnou x-ovou souřadnici. Stačí si uvědomit, že potřebujete pouze čtyři jednoduché rovnosti, aby si METAFONT uvědomil, že obrazec složený z bodů *1*, *2*, *3* a *4* je obdélník:

$$y_1 = y_2;\ y_3 = y_4;$$

$$x_1 = x_4;\ x_2 = x_3;$$

Odteď METAFONT ví, že obraz složený z bodů *1*, *2*, *3* a *4* je obdélník. Nakonec stačí už jen určit souřadnice dvou protilehlých bodů (třeba *1* a *3*, nebo *2* a *4*) a celý obdélník je perfektně definován.

Další možností, je udat vzdálenosti. Pokud by jsme chtěli, aby vzdálenost mezi body *1* a *2* byla 100 pixelů, stačí zápis:

$$x_1 = x_2 + 100;\ \text{nebo}\ x_1 - x_2 = 100;$$

Co se týče výšky, pokud budeme chtít, aby bod *3* byl vždy dvakrát výše než bod *2* zapíšeme to takhle:

$$y_3 = 2y_2;$$

Nebo si dokonce můžete udělat souřadnice bodů závislé na nějaké parametru, který si můžete nadefinovat jak chcete. Například:

```
x1 = 1/2a;
```

To znamená, že x-ová souřadnice bodu *1* je rovna polovině hodnoty parametru „*a*“. Jak je vidět, díky algebře je výčet možností téměř nekonečný.

## 3.9 Vektory

Nyní už víme, že můžeme x-ové i y-ové souřadnice sčítat, „$x_1+x_2$“ a „$y_1+y_2$“ a máme obě definované. Takže nebude velkým překvapením, že můžeme definovat „$(x_1, y_1) + (x_2, y_2)$“ jako „$(x_1 + x_2, y_1 + y_2)$“. Touto operací jsme definovali sčítací operaci pro dvojici souřadnic (a od zápisu „$z_1$“, který je stejný jako „$(x_1, y_1)$“) můžeme také psát „$z_1+z_2$“. Co ale tyto operace znamenají?

Odpověď dostaneme, pokud se vrátíme k naší definici souřadnic. Řekli jsme, že souřadnice bodu jsou dvě čísla udávající vzdálenost napravo a nahoru od referenčního bodu. Protože při této definici jsou souřadnice referenčního bodu vždy (*0, 0*). Nyní si ale vezměme bod o souřadnicích (*x, y*). Přidáním nuly se nám na nich nezmění a my můžeme říct, že souřadnice jsou (*0 + x, 0 + y*) a díky definici páru popsané dříve můžeme psát (*0, 0*) + (*x, y*). To vypadá povědomě, ne? Ano, vidíme referenční bod a po přidání náhle vypadá docela jako popis souřadnic, který je na začátku tohoto odstavce. Nyní, když víme, že přidání jisté vzdálenosti k x-ové souřadnici bodu znamená „pohyb vpravo o tuto vzdáleností“, a že přidání jisté vzdálenosti k y-ové souřadnici bodu znamená „pohyby nahoru o tuto vzdálenost“. Takže přidání páru jednoduše znamená přidání obou přírůstků ve stejnou dobu. Tím se oba posunují vpravo a nahoru oběma specifikovanými vzdálenostmi. Jinak řečeno, páry souřadnic

mohou popsat nejen pozici, ale také posunutí. Záležet na tom, co uděláme. Pár (*5, 10*) může znamenat „bod umístěný v 5 jednotkách vpravo a 10 jednotkách nahoře od referenčního bodu“, nebo „pohyb vpravo o 5 jednotek a nahoru o 10 jednotek“. Tento druhý význam se uplatní, jakmile přidáme tento pár k dalšímu páru souřadnic. Nyní, když už víme něco o rovnici, jako je „*z 2 = z 1 + (5, 10)*“, znamená to jednoduše řečeno: jdeme do bodu 2 z bodu 1 tak, že se pohneme o 5 jednotek vpravo a o 10 jednotek nahoru.

Posunutí definovaná pomocí páru souřadnic budeme nazývat vektory. Vektor můžeme zakreslit jako šíp ukazující, kam se posun vede, a jehož délka je vlastně množství posunutí. Vektory a body jsou kompletně ekvivalentní, protože souřadnice bodu jsou také posunutí nezbytná k posunu z referenčního bodu do bodu, o který nám jde. Všimněte si také, že stejně jako body mohou mít záporné souřadnice, mohou i vektory mít záporné souřadnice. Ty nám říkají, že se jedná o posun doleva a dolů. Dvojice (*0, 0*) je také vektor. Tento vektor se ale nikam nepohybuje a nazývá se nulový vektor.

## 3.10 Rozšíření algebraických pojmů

Než se začneme zabývat dalšími algebraickými operacemi v METAFONTu, uděláme si v nich trochu pořádek. Rozdělíme si je do tří kategorií podle toho, na co se operace dají použít:

1) operace použitelné na numerické hodnoty
2) operace pro páry hodnot
3) operace související s úhly

Toto rozdělení není úplné, nějaké operace jako „*sqrt*“ jdou aplikovat pouze na numerické hodnoty, jiné operace mohou být aplikována pouze na páry hodnot, ale část operací (jako závorky) mohou být aplikovány na více kategorií (jak na numerické operace, tak i na operacích s páry).

### 3.10.1 Numerické operace

Než si začneme povídat o numerických operacích. Podíváme se na tři speciální numerické proměnné, které jsou v METAFONTu definovány. První, o které se zmíním, je „*epsilon*“, která odpovídá nejmenší kladné hodnotě, která však není rovná nule, a se kterou je METAFONT schopen pracovat. Její hodnota je 1/65536. Druhou je „*infinity*“, která naproti svému jménu nekonečná není. Definice „*infinity*“ je rovna 4096 – „*epsilon*“. Poslední proměnnou je „*whatever*“, která se používá, pokud nechceme definovat přesnou hodnotu.

Již jsme viděli, že „*sqrt*“ bere druhou odmocninu z numerické hodnoty. Jak ale udělat umocnění? V METAFONTu se pro umocnění používá operátor „**“. Takže zápis „*a**b*“ znamená *a* na *b*. Takže čtverec z „*a*“ uděláme zápisem „*a**2*“, pokud bychom chtěli krychli, pak zápis je „*a**3*“.

METAFONT používá operátor „/“ pro přesné dělení. Někdy ale potřebujeme celočíselné dělení, pro to existuje operátor „*div*“. Zápis „*a div b*“ vrátí celé číslo. V METAFONTu existuje ještě jeden operátor spojený s dělením a to operátor „*mod*“, který vrací zbytek po celočíselném dělení.

Dalšími praktickými operátory jsou „*abs*“, „*max*“ a „*min*“. Operátor „*abs*“ vrací absolutní hodnotu čísla, „*abs (-7)*“ je 7. Operátor „*max*“ bere seznam hodnot a vrací z něj největší hodnotu, stejně, ale opačně pracuje operátor „*min*“, který vrací nejmenší hodnotu ze seznamu. Příklady: „*max (2, 5, 8)*“ vrátí 8, zatímco „*min (2, 5, 8)*“ vrátí 2.

A nakonec, navzdory skutečnosti, že METAFONT pracuje i s necelými čísly, někdy budete potřebovat je zaokrouhlit. Pro zaokrouhlování má METAFONT tři operátory: „*floor*“, „*ceiling*“, „*round*“. Jejich význam je uveden na obrázku:

| | floor | ceiling | round |
|---|---|---|---|
| 3.1416 | 3 | 4 | 3 |
| −3.1416 | −4 | −3 | −3 |
| 1.5 | 1 | 2 | 2 |
| 1.49 | 1 | 2 | 1 |

obrázek 7

Popis chování zaokrouhlovacích operátorů

„*floor*“ zaokrouhluje číslo na nejbližší nižší celé číslo. „*ceiling*“ zaokrouhluje na nejbližší vyšší celé číslo a „*round*“ zaokrouhluje na absolutně nejbližší celé číslo, v případě, že se má pomocí „*round*“ zaokrouhlovat číslo, které je přesně mezi dvěma čísly (třeba 2.5, nebo –1.5), zaokrouhlí se na nejbližší vyšší celé číslo.

### 3.10.2 Operace pro párové hodnoty

Stejně jako pro numerické hodnoty obsahuje METAFONT i několik praktických operátorů pro dvojice hodnot. Jsou jimi: „*up*“, „*down*“, „*left*“, „*right*“ a „*origin*“. Jejich příslušné hodnoty jsou (0, 1),(0, -1),(-1, 0),(1, 0) a (0, 0). Operátor „*origin*“ vždy odkazuje na referenční bod a zbylé nejlépe popíšeme jako vektory indikující čtyři nejjednodušší směrnice (přidáním „*up*“ k páru souřadnic znamená „pohni se o jeden pixel nahoru“).

Existuje operátor „*t[$z_1$, $z_2$]*“, který pracuje s párem proměnných jako s body (vrací souřadnice bodu uspořádaného se dvěma body a považuje ho za operand). Další důležitou operací, která bere pár souřadnic jako vektor je operátor „*length*“. Tento operátor vrací délku (numerickou hodnotu) vektoru. Jeho praktický význam je v tom, že nám umožňuje zjisti vzdálenost mezi dvěma body. Pamatujte si, že když máte dva body 1 a 2, $z_2 – z_1$ je vektor spojující tyto dva body, takže „*length($z_2$ - $z_1$)*“ vrátí vzdálenost mezi těmito

body. To je velmi užitečné, pokud chcete definovat bod podle jeho vzdálenosti k jinému bodu.

Již jsme viděli, je možné definovat produkt vektoru číselnou hodnotou. Je ale možné definovat produkt mezi vektory? Ve skutečnosti ne. Tady je produkt takový, že výsledkem je spíše číselná hodnota než další vektor. Takový produkt je nazýván skalárním součinem, který obvykle zapisovaný tečkou a má následující definici:

$$(a, b) \cdot (c, d) = ac + bd$$

V METAFONTu se skalární součin zapisuje pomocí operátoru „*dotprod*“, který se umísťuje mezi jeho operandy. Jaké využití může mít taková divná definice? Stává se, že definice skalárního součinu je rovna 0, potom jsou tyto produkty ortogonální, to znamená, že jsou na sebe kolmé. S tímto produktem můžete snadno definovat kolmé čáry. Přestavte si, že máte definovanou čáru mezi dvěma body 1 a 2 a chcete nadefinovat jinou čáru, která bude procházet přes body 3 a 4 a bude na první čáru kolmá. Vše, co potřebujete k definici druhé čáry je zápis „$(z_2 - z_1)$ *dotprod* $(z_4 - z_3) = 0$“ a máte pak jistotu, že mezi čárami bude pravý úhel.

### 3.10.3 Úhlové operace

Všechny operace, kterými jsme se dosud zabývali používají číselné hodnoty jako je délka a vzdálenost. V geometrii jsou však další důležité číselné hodnoty a těmi jsou úhly a mnoho geometrických operací pracuje radši s úhly než vzdálenostmi.

V METAFONTu jsou úhly číselné hodnoty. Nejsou formálně odlišné od délkových hodnot. Ale operátory očekávající úhly, pracují s jejich hodnotami jinak. Hlavní rozdíl je, že METAFONT pracuje s úhly ve stupních, takže číselná hodnota použitá jako úhel bude brána ve stupních a ne v pixelech. To

předpokládá, že si nebudete plést úhly s délkovými jednotkami. Proto byste neměli používat stejné proměnné pro úhlové hodnoty a délkové hodnoty současně. Jinak si v kódu uděláte akorát zmatek, a to jak pro sebe, tak i pro každého čtenáře vašeho kódu.

Základní operátory pro úhly jsou samozřejmě sinus a cosinus. Jejich názvy jsou „*cosd*“ a „*sind*“. V METAFONTu je na konci “d“ , abychom nezapomněli, že úhly v METAFONTu jsou měřeny ve stupních (anglicky degress), a ne třeba v radiánech. Pravděpodobně je však nikdy nevyužijete v METAFONTovém programu. Protože METAFONT poskytuje dva další operátory, které budete spíš používat pro návrh fontu. Tyto operátory se nazývají „*dir*“ a „*angle*“. „*dir*“ bere jako hodnotu úhel, a vrací vektor o délce 1, který je natočen horizontálně v úhlu, který jste definovali (všimněte si, že úhly v METAFONTu jsou měřeny kladně a proti směru hodinových ručiček. Takže 0 odpovídá horizontální poloze směřující doprava. To znamená, že kladný úhel, mezi 0 a 180 pak vektor povede nahoru, a dolů povede se záporným úhlem). Například, operátor „*up*“ může být definován jako „*dir90*“. „*angle*“ je podobná operace: bere vektor jako operand, a vrací úhel tohoto vektoru vůči horizontále. Kladné úhly jsou opět proti směru hodinových ručiček. Například výraz „*angle up*“ bude vracet 90.

## 3.11 Transformace

Dosud jsme mluvili pouze o algebraických operacích, operacích aplikovaných na čísla, páry. Transformace jsou trochu o něčem jiném. Transformace jsou geometrické operace, jejich účelem je měnit tvary a nebo pozice obrazců. Jako takové nemohou být aplikovány na číselné hodnoty, ale jen na páry hodnot, nebo tam, kde se dají považovat za vektory. Transformace mohou být popsány jako algebraické operace, ale to je pouze zápis. Transformace mají hlavně geometrickou definici, a tak si je i představíme.

Protože METAFONT se při práci s nimi soustředí na jejich geometrický význam a ne na jejich algebraické detaily.

Všechny transformace v METAFONTu mají stejnou syntaxi: „párová hodnota transformovaná nějakým parametrem (s)“. To znamená, že transformace se vždy umísťuje za párovou hodnotu, na kterou má být aplikovaná.

Základní METAFONTové transformace jsou „*shifted*“ a „*scaled*“. „*shifted*“ bere jako parametr vektor a po použití na pár hodnot, posune ho o množství definované vektorem. To je úplně stejné jako sčítání párů. Když máme „*(x, y) shifted (a, b) = (x + a, y + b) = (x, y) + (a, b)*“. „*scaled*“ bere jako parametr číselnou hodnotu a touto hodnotou se násobí obě souřadnice páru. Je to stejné jako násobení páru hodnot nějakým číslem: „*(x, y) scaled s = (sx, sy) = s(x, y)*“. Důvod proč psát raději transformace než jednoduché operace je čistší syntaxe, díky které snáze pochopíme o jakou geometrickou transformaci se jedná.

„*scaled*“ násobí obě souřadnice páru stejnou hodnotou. Je efektivní, pokud chceme obrazec scalovat o stejný koeficient jak horizontálně, tak i vertikálně. Někdy však chceme obrazec natáhnout nebo smrštit pouze horizontálně, nebo vertikálně a nebo každým směrem jiným koeficientem. Pro tyto účely má METAFONT transformace „*xscaled*“ a „*yscaled*“. Fungují stejně jako „*scaled*“, ale aplikují se pouze na jednu souřadnici z páru, podle svého názvu. Takže „*(x, y) xscaled s = (sx, y)*“ a „*(x, y) yscaled s = (x, sy)*“. Pokud chceme aplikovat dva různé koeficienty na souřadnice páru zároveň stačí zapsat nejprve jednu transformaci a poté druhou: „*(x,y) xscaled s yscaled t = (sx, ty)*“.

Díky operacím „*scaled*“ a „*shifted*“ již můžeme dokázat hodně, ale stále budeme trochu omezeni, proto si řekneme něco o rotacích. Rotace spočívá v otočení obrazce kolem nějakého bodu, zatímco obrazec si všechny své

rozměry ponechá stejné. Rotace je jednou z nejdůležitějších a nepraktičtějších transformací, které METAFONT nabízí. METAFONT nám nabízí dvě instrukce pro rotace. První a jednodušší je „*rotated*“, ta si bere jako parametr úhel a znamená: „otoč bod okolo referenčního bodu (0, 0) o daný úhel“. Druhou instrukcí je „*rotatedaround*“. Tato instrukce potřebuje dva parametry. Prvním je dvojice a druhým je úhel. Parametry musejí být odděleny čárkou a zapsány do závorek. Příklad: „*rotatedaround(*$z_1$*, 90)*“. Úhel má stejný význam jako v předchozím případě. Dvojice nám určuje souřadnice bodu, kolem kterého se bude rotovat. Jinak řečeno „*rotated*“ je vlastně speciální případ „*rotatedaround*“, který by se dal zapsat jako „*rotatedaround((0, 0), úhel)*“.

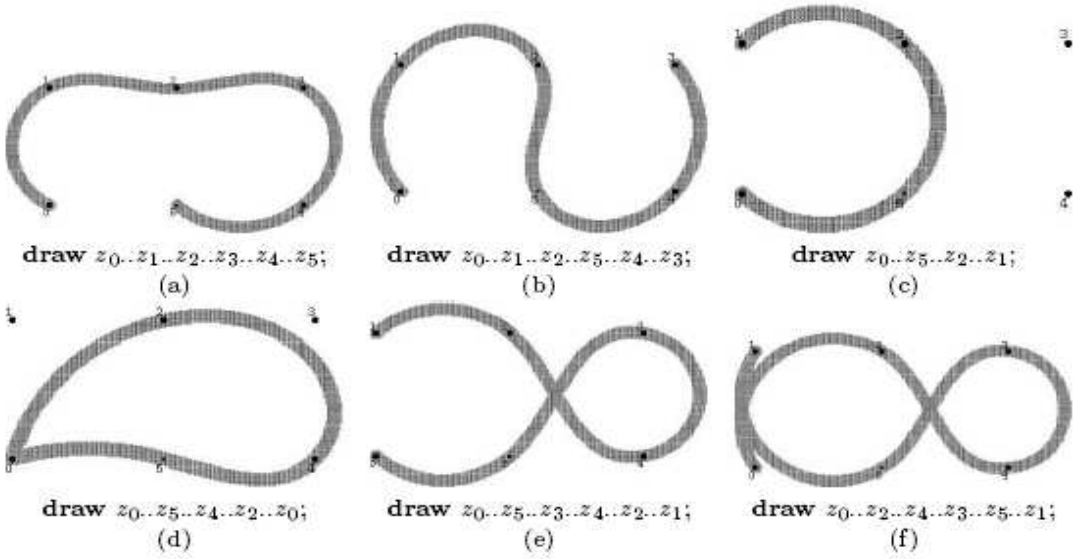
Poslední důležitou transformací, která je velmi užitečná při tvorbě zrcadlově souměrných objektů je „*reflection*“, která spočívá v tom, že vezme bod, který je přímo naproti dalšímu bodu na druhé straně osy. To je velmi důležité, neboť výraz souměrný se obvykle používá při odvolávání se na zrcadlení a objekty se nazývají souměrné, když mají stejný tvar na obou stranách své osy. K tomuto účelu nám METAFONT poskytuje transformaci „*reflectedabout*“, která bere jako parametry dva páry proměnných, které reprezentují dva body na čáře určené k zrcadlení. Tyto dvě proměnné musí být odděleny čárkou a v závorkách. Příklad: „*z1 reflectedabout (z2, z3)*“.

## 3.12 Křivky

Nerovné čáry se jednoduše nazývají křivky a jsou to čáry, které nejsou rovné. Dosud jsme se setkali jen s kreslením rovných čar, pomocí příkazu „draw“ a souřadnic koncových bodů čáry, které byly odděleny „..“. Příklad křivky:

```
draw (20, 45)..(50, 35)..(85, 40);
```

Jak je vidět na příkladu, příkaz „*draw*“ je následován třemi body a ne dvěma. Souřadnice jsou opět odděleny „..“. To je tajemství síly příkazu „*draw*“ v METAFONTu. Za příkaz „*draw*“ můžete napsat jakékoliv souřadnice, vždy oddělené „..“ a METAFONT vykreslí vždy hladkou křivku, která bude procházet přes zadané body. Ukázka křivek v METAFONTu:




obrázek 8

Ukázka křivek

Jak je vidět na příkladech, METAFONT někdy ohýbá křivky docela hodně, je to proto, aby dostal co nejhladší výsledek. Je to proto, že, jak je užívaný právě teď, bude vždy konstruovat křivky tak, aby ohýbání bylo minimální v zadaných bodech (to si můžete ověřit na ukázkách křivek, když jde křivka mimo jeden ze zadaných bodů, udělá se přímo, jak jen to je možné). Další pravidlo, které METAFONT používá, když kreslí křivky je, že mezi dvěmi následnými body danými v příkaz „*draw*“ se bude pokoušet nakreslit křivku bez ohýbání, to jest křivku, která se ohýbá jen jedením směrem. METAFONT poruší toto pravidlo jen když nemůže žádným další způsobem udržet křivku hladkou. To vysvětluje, proč v příkladu (f), METAFONT

nakreslil taková velká oka mezi body 0 a 2, a mezi body 5 a 1. Více přímé křivky mezi body můžete vytvořit pomocí ohýbacího bodu, a to je i nejlepší způsob, jak držet hladkou křivku (z pohledu METAFONTu). Na druhé straně, METAFONT nemá žádný problém s vytvořením ohýbacího bodu v pozici bodu daného v příkazu „*draw*“. Podívejte se například na bod 5 z příkladu (d). Pokuste se zapamatovat si tyto dvě pravidla pro kresbu. Pomůžou vám správně zapsat křivku, kterou chcete, aby METAFONT vykreslil.

„*draw*“ instrukce je velmi silná. Nicméně, jestliže jste zkoušeli sami nakreslit několik křivek, tak jste si pravděpodobně všimli, že je někdy až příliš silný, ve smyslu že METAFONT dělá rozhodnutí o tvaru křivky sám a někdy nevykreslí přesně to, co máte na mysli. To je ale logické: zadání pozice několika bodů je docela malá informace, a METAFONT nemůže číst vaši mysl! Z pozic bodů, které jste zadali, se bude pokoušet udělat pěkně hladkou křivku, ale ta bude podle jeho vlastních kritérií, protože nemá žádný způsob jak odhadnout vaše kritéria. To má za následek, že instrukce „*draw*“ někdy vykreslí něco jiného, než jste očekávali.

Takže to znamená, že METAFONT je šéf a my musíme akceptovat jeho rozhodnutí a doufat, že někdy vykreslí obrazce, tak jak chceme my? Samozřejmě, že ne! METAFONT je jen počítačový program a udělá přesně to, co chceme, jen mu musíme říct, aby poslouchal nás. Jinými slovy, když budeme chtít, aby METAFONT vykreslil křivku přesně tak jak chceme, budeme mu muset někdy dát víc informací, než jen pozice bodů. Jak to udělat? Nyní si řekneme o dvou možných postupech, jak to udělat.

Jednou z možností, jak kontrolovat obrazec křivky, je specifikovat její směr, nebo zkosení v nějakém bodu, nebo bodech. METAFONT dovoluje specifikovat směr křivky, když půjde kolem bodů, které používáte k definici vaši křivky (jinak řečeno, za účelem řízení sklonu křivky v nějaké pozici, potřebujete souřadnice této pozice zapsat do příkazu „*draw*“). Specifikovat směr je samo o sobě docela snadné, když si pamatujete, že souřadnici páru

nedefinují pouze pozice bodů, ale můžou se také využít pro definici vektorů (posunutí). Posunutí se vždy realizuje podle určitého směru, použití vektorů pro určení směrnice je přirozené. Ke specifikaci směru v nějakém bodu pojmenovaném ve vašem „*draw*“ příkazu potřebujete jen dát souřadnice páru, který bude interpretován jako vektor, v složených závorkách „*{}*“, a položit celý zápis vedle „*z*“ (nic nemůže být mezi „*z*“ a vektorem, zvláště ne „..“). Ukázka této syntaxe:

```
draw <something>..{(a, b)}z..<something else>;
```

nebo

```
draw <something>.. z{(a, b)}..<something else>;
```

zde (a, b) jsou souřadnice páru definované jako vektor. Nemá cenu si dělat starosti s pozicí {(a, b)} části. Nezmění se nic, pokud tento zápis bude před a nebo za souřadnicemi bodu.

Můžete si myslet, že to je skvělé, ale definice směrnice párem souřadnic není tak jednoduchá. Na první pohled pravděpodobně nemůžete vidět, který směr odpovídá vektoru o souřadnicích (-5.8, 12.3) a jaký vektor odpovídá směru pod úhlem 13.5° směřující doleva a nahoru z horizontály. Je na čase, připomenout si podsekce 2.10.2 a 2.10.3, kde byli uvedeny série předdefinovaných hodnot operátorů, které jsou zde, aby se postaraly o tuto práci. Pokud potřebuje jednoduchý směr, stačí použít vektory „*up*“, „*down*“, „*left*“ a „*right*“, jejichž jména korespondují přesně se směrnicemi, které definují. Pamatujte si také, že jestliže máte dva body označené 1 a 2. $z_2 - z_1$ je vektor odpovídající posunutí mezi bodem 1 a bodem 2, směrem k bodu 2. Můžete toho využít, pokud chcete křivku vést ve směru definovaném těmito body. A nakonec, jestliže znáte úhel směru, kterým chcete křivku vést, neváhejte použít „*dir*“ k vytvoření jednotkového vektoru definujícího tento směr. Nyní již máte všechny nezbytné nástroje k definování směrnice bez přesné znalosti, jak souvisejí souřadnice vektoru se směrem, který definuje.

Nyní jste schopni kontrolovat tvar vaší křivky, ale jste stále omezeni faktem, že jakákoliv křivka, kterou vytvoříte bude otevřená. Dokonce i když uděláte křivku, která bude začínat a končit ve stejném bodě viz obrázek 8 (d), výsledná křivka nebude úplně hladká. To je způsobeno tím, že se METAFONT nestará o to, jestli jste při kreslení prošli jedním bodem vícekrát. Takže jak vykreslit uzavřené tvary jako jsou O, nebo 8?

Asi by vás napadlo, poté co již víme, stanovit směr křivky v bodě, kde začíná a končí. Příklad tohoto zápisu (upravený kód z obrázku 8 (d)):

```
draw z0{right}..z5..z4..z2..{right}z0;
```

Výsledkem bude křivka, která bude vypadat uzavřeně a hladce. Problém tohoto přístupu je ten, že ovládáte směr křivky z jejího výchozího bodu, což je něco, co nesmíte dělat (nesmíte dokonce ani umět popsat směr křivky, který by měla mít). Spíše by jste měli postupem „pokus, omyl“, nalézt úhel a použít příkaz „dir“, ale to by bylo špatné programování. Zvláště pokud se pak rozhodnete změnit pozice nějakých bodů křivky, pak budete muset pravděpodobně změnit i úhel, znovu pomocí taktiky „pokus, omyl“. Samozřejmě byste mohli vždy vypočítat směr, ale to by často chtělo hodně práce, a někdy by to ani nebylo možné udělat.

Sice jsem důkladně vysvětlil, proč je předchozí přístup špatný a nepraktický, ale nedal jsem žádnou jinou alternativu. METAFONT nabízí jednoduchý způsob, jak to udělat. Namísto uvažování, že projdeme dvakrát přes stejný bod, na začátku a na konci příkazu „*draw*“, nahradíme druhé zapsání počátečního/koncového bodu slovem „*cycle*“. Přítomnost tohoto slova bude METAFONTu říkat, že chceme nakreslit uzavřenou křivku a METAFONT automaticky spojí poslední bod zmíněný před slovem „*cycle*“ s prvním bodem křivky v hladké cestě. Příklad použití „*cycle*“:

```
draw z0..z5..z4..z2..cycle;
```

## 3.13 Pera

K vysvětlení, jak METAFONT ovládá grafiku použiji metaforu, totiž metaforu kresby. METAFONT není postava za stolem a s archem papíru před sebou. Je to program, a všechno, co dělá, když manipuluje s grafikou, je, že ovládá mřížku složenou z pixelů, malé jednotky prostoru, která má jen dva stavy: bílý nebo černý. METAFONT dělá jen to, že říká který pixel bude černý a který zůstane bílý. Metafora kresby je velmi užitečná, protože z naší perspektivy to vypadá přesně jako kdyby METAFONT byl tou postavou za stolem a na kus papíru kreslil, co mu přikážeme kreslit. Není tomu jen tak pro nic za nic, že hlavní kreslící instrukce se nazývá „*draw*“. Stejně, jako když kreslíme ručně, tak i METAFONTové křivky se chovají jako by byly skutečnými tahy v určité smyslu.Vedou z jednoho bodu do dalšího ( pořadí je samozřejmě stejné jako pořadí bodů zadané instrukcí „*draw*“).

Tato metafora kresby může být ještě rozšířena. Když chcete něco nakreslit, nepotřebujete jen vaši ruku a kus papíru, ale také pero, bez kterého byste nikdy na papír nic nenakreslili. A v závislosti na druhu pera, které používáte (tužku, kuličkové pero, plnicí pero, nebo jedno z kaligrafických per), dostanete různé výsledky, dokonce i když tvar, který kreslíte, je pokaždé stejný. V METAFONTu stačí udělat přesně to samé. METAFONT má příkaz „*pickup*“, který vám dovolí vybrat si pero a to budete používat pokaždé, když zavoláte instrukci „*draw*“. Použitím tohoto příkazu si můžete vybrat určité pero, které je malé nebo velké, méně nebo více kruhové (nebo nějakého dalšího tvaru) atd. Příkaz „*pickup*“ musí být následovaný jménem pera, které chcete použít.

K vybrání pera, ale potřebujete nějaká pera, ze kterých je možno si zvolit. To není žádný problém, protože METAFONT má několik předdefinovaných per připravených k použití, kromě standardního pera, které

se automaticky používá, když si nevyberete jiné, jsou dvě hlavní předdefinovaná pera, která většinou použijete. Nazývají se pencircle a pensquare. Jejich názvy korespondují s jejich tvarem k peru. Pero tvořené kruhem o průměru 1 pixel a nebo pera čtvercového typu o straně 1 pixel.

Malinká pera mohou být užitečná, ale obvykle chceme větší pera. Jak můžete dostat taková pera, když máme jen pera velmi malého typu? Odpověď je vlastně docela jednoduchá: transformovat je! Všechny transformace, které jsou uvedeny v části 2.10.2 můžete použít nejen na párové hodnoty, ale i na pera! Nebo přinejmenším, transformace, které mají smysl v této situaci. Pokud je naším cílem změnit tvar a nebo velikost našeho pera, transformace jako „*shifted*“, „*rotatedaround*“ nebo „*reflectedabout*“ se docela hodí. „*scaled*“ ( změní velikost našeho pera), „*xscaled*“ a „*yscaled*“ ( natáhnout naše pero jedním směrem) můžou být také užitečné při práci s pery. Příklady per:

```
pickup pencircle scaled 10;
```

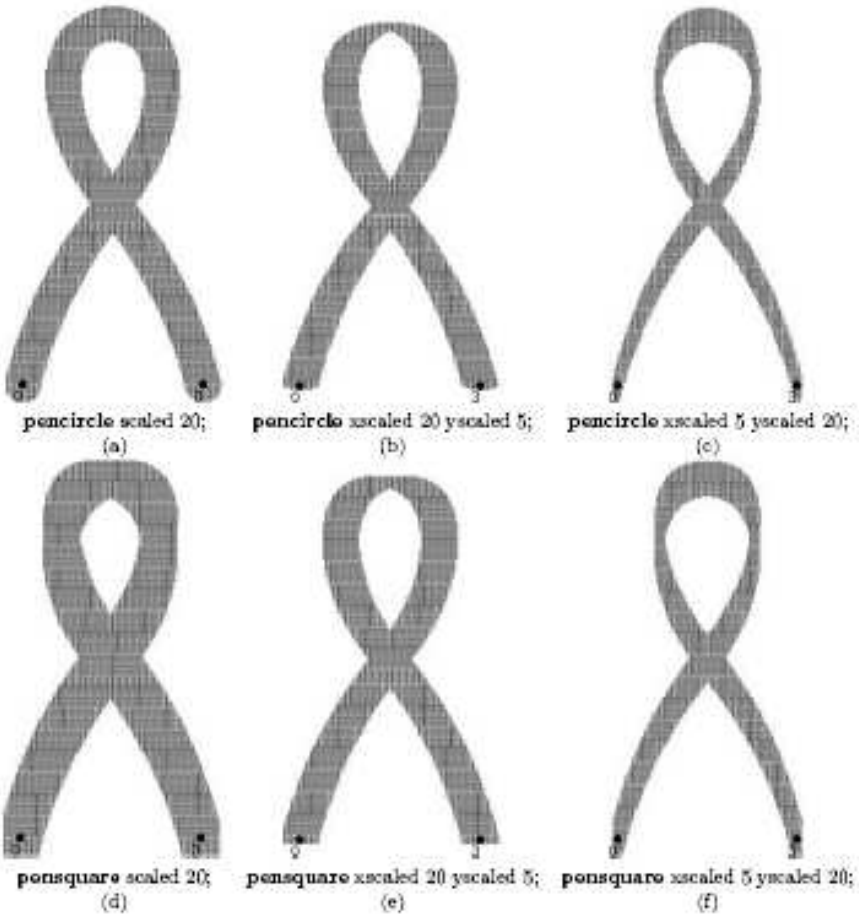
vytvoří pero kruhového typu o průměru 10 pixelů

```
pickup pensquare xscaled 20;
```

získáte pero pravoúhlého typu široké 20 pixelů a 1 pixel vysoké a můžete také použít transformace tohoto typu:

```
pickup pencircle xscaled 10 yscaled 35;
```

které udělají naše pero jako elipsoid, s horizontální osou dlouhou 10 pixelů a svislou osou dlouhou 35 pixelů.

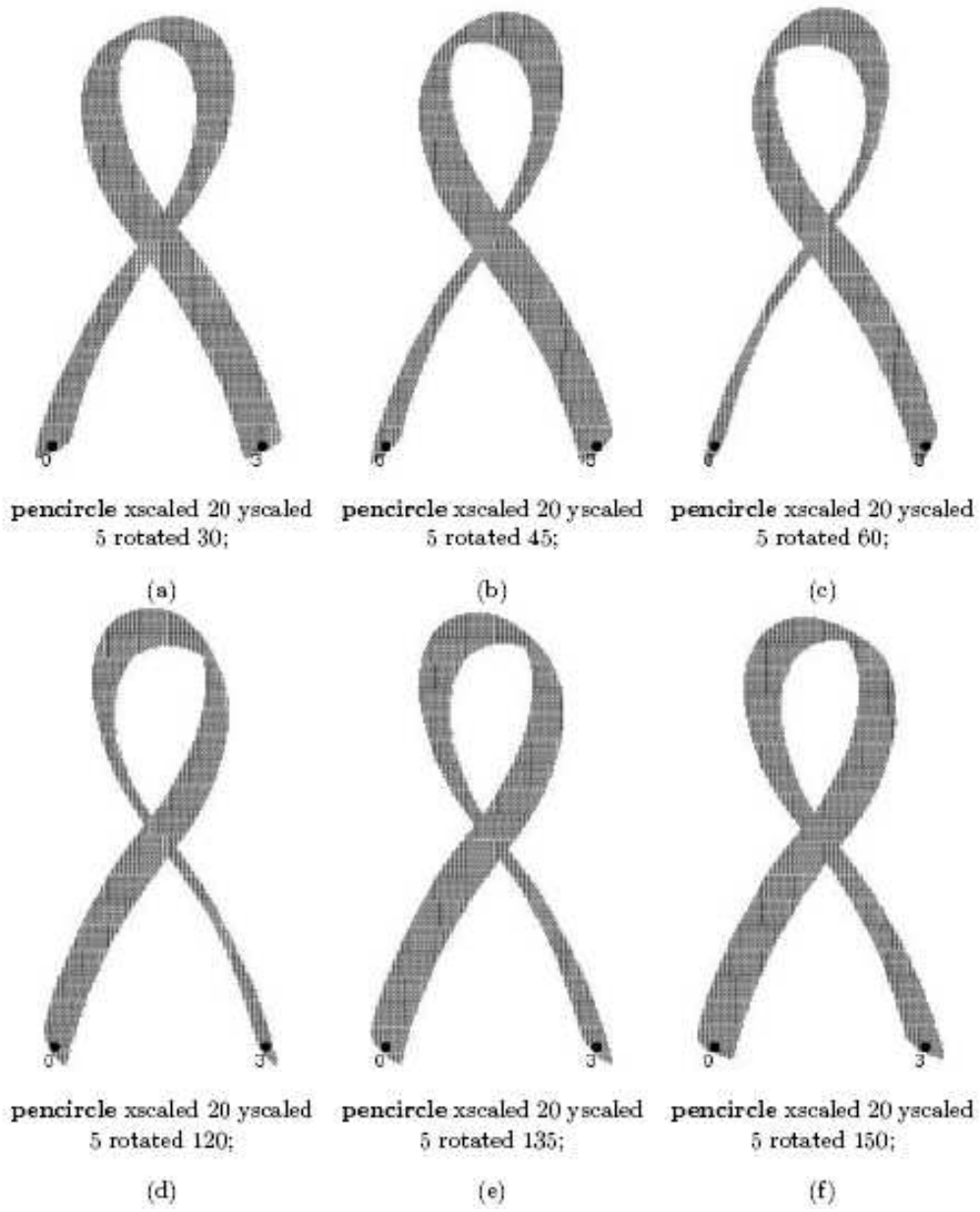



obrázek 9

Pera v METAFONTu

Nicméně, jestli je toto všechno, jak můžete upravit velikost „*pencircl*e“ a „*pensquare*“, pak budete rapidně omezení v tom, co můžete udělat. Dokonce když upravíte míry pera různými faktory vodorovnými a svislými, vždy dostanete buď horizontální nebo svislá pera. Jestliže víte něco o kaligrafii, pak víte, že to, co nejvíce ovlivňuje vzhled písma, kromě tvarů křivek, je pozice tenkých a tlustých čar. A to, co stanoví jejich pozici, je úhel, který hrot pera svírá s horizontálou. Kaligrafové točí svoje pera v závislosti na tom, čeho chtějí dosáhnout. A to je něco, co METAFONT může snadno udělat díky rotaci jeho per. Ačkoliv transformaci „*rotatedaround*“ nemůžete použít s pery, „*rotated*“

transformace může a má za následek, že je hrot pera našikmo k horizontále. Obrázek ukazuje, jaký účinek má samotný úhel na vzhledech pera u jinak stejných křivek:




obrázek 10

Upravená METAFONTová pera

## 3.14 Podmínky

Syntaxe METAFONTovské podmínky je následující:

```
if podm1:text1 ;
elseif podm2:text2
elseif podm3:text3 ;
...
else:text4 ;
fi;
```

Povinná je pouze část na prvním řádku začínající „*if*“ a ukončení příkazu „*fi*“, ostatní části se uvádět nemusí. METAFONT postupuje tak, že nejprve vyhodnotí podmínku za „*if*“ , když je splněna, provede text uvedený za ní. Pokud splněna není, pokračuje částmi „*elseif*“. Ty se vyhodnocují v pořadí, v jakém byly uvedeny. V případě, že není splněna ani jedna podmínka, provede METAFONT text za „*else*“. Chceme-li například spojit bod $z_3$ s nejbližším z bodů $z_1$ a $z_2$, a pokud jsou stejně daleko, nakreslit obě spojnice, napíšeme:

```
if length(z1-z3)<length(z2-z3): draw z3--z1;
elseif length(z2-z3)<length(z1-z3): draw z3--z2;
else: draw z3--z1;
draw z1--z2;
fi;
```

Relační operátory METAFONTu jsou: >, >=, <, <=, = a <>. Jako logické operátory se kromě klasických and, or a not používají ještě numeric, path, boolean, string, pen, picture, transform a pair. Tyto operátory mají hodnotu true

jen tehdy, pokud je jejich operand daného typu, tedy path p=true jen když p je typu path.[5]

## 3.15 Cykly

Cyklus je část textu, která se pro různé hodnoty tzv. řídící proměnné vyhodnotí několikrát. METAFONT užívá několik druhů cyklu. První z nich má syntaxi:

```
for prom = hodn1 , hodn2 ,. . .:text endfor
```

Tento zápis METAFONT vyhodnotí tak, že do proměnné „*prom*“ dosadí hodnotu hodn1 a proběhne tělem cyklu text, poté dosadí hodnotu hodn2 a proběhne podruhé atd. Pokud např. potřebujeme spojit bod $z_0$ s body $z_3$, $z_4$ a $z_7$, můžeme napsat cyklus:

```
for i=z3,z4,z7:
  draw z0--i;
  endfor;
```

Hodnoty řídící proměnné často tvoří aritmetickou posloupnost. V takovém případě použijeme cyklus:

```
for prom = dolni step krok until horni : text
                    endfor
```

Pokud je krok 1 resp. −1 můžeme místo step _1 until napsat pouhé upto resp. downto. Tento druh cyklů se většinou využije v různém rastrování a generování pravidelných vzorů. Milimetrový papír o rozměrech 1*x*1 cm vygenerujeme cyklem:

```
for i=0 step 1mm until 1cm:
    draw (i,0)--(i,1cm);
    draw (0,i)--(1cm,i);
```

```
endfor;
```

Tělo cyklu, text, nemusí být vždy kompletním příkazem zakončeným středníkem. V případě, že je text pouze částí příkazu, středník za ním nepíšeme. Poslední typ cyklu, který zde uvedu, je nekonečný cyklus:

```
forever:text endfor
```

Takovýto cyklus nemá konec, vyskočit z něj lze pouze příkazem „*exitif*“. Tento příkaz může být uveden v těle jakéhokoli cyklu; pokud je podmínka za ním splněna, METAFONT okamžitě přeruší zpracování tohoto cyklu a pokračuje textem za endfor. [5]

## 3.16 Přehled důležitých METAFONTových příkazů

Nakonec kapitoly o programu METAFONT jsem se rozhodl umístit přehled příkazů METAFONTu. Nejedná se o úplný přehled, ten je k nalezení v knize *The METAFONTbook* od D. E. Knutha. Uvedené příkazy patří mezi nejdůležitější:

**&** - draw p1 & p2; volné navázání cest p1 a p2
**[]** - z3=.5[z1,z2]; lineární kombinace bodů - bod z3 leží v polovině mezi z1 a z2
**--** - z1--z2; rovná čára
**---** - z1---z2; křivka s maximálním napětím
****** - c=a**b; umocňování
**..** - z1..z2; volná křivka
**...** - z1...z2; křivka s napětím větším než 1
**abs** - abs z1; absolutní hodnota vektoru
**and** - logický součin
**angle** - angle(z1); úhel vektoru (-180ř,180ř)
**beginchar** - beginchar("A",20pt#,30pt#,0); začátek definice znaku, parametry:

umístění znaku, šířka, výška, hloubka

**boolean** - boolean l; logická proměnná

**ceiling** - ceiling a; zaokrouhlování nahoru

**controls** - z1..controls z2 and z3..z4; křivka s pomoc. body

**cosd** - cosd40; =cos 40ř

**cullit** - cullit; záporné hodnoty bodu nahradí 0, kladné 1

**cycle** - z1..z2-z3..cycle; uzavření cesty

**def** - def velkybod(expr bod)= počátek definice makra

**define_pixels** - define_pixels(perosl,perotl); přepočet jednotek na pixely

**dir** - dir20; jednotkový vektor svírající úhel 20ř s osoy *x*

**direction** - direction t of p; směr cesty p v čase t

**down** - draw z1down..z2; bod (0,-1)

**draw** - draw p; nakreslení cesty p

**drawdot** - drawdot z1; vykreslí bod zvoleným perem

**end** - end; konec souboru

**endchar** - endchar; konec znaku

**endfor** - endfor konec cyklu for

**epsilon** =1/65536 nejmenší číslo větší než nula

**erase** - erase draw p; odmaže cestu p

**expr** - def ... (expr a)= parametr libovolného typu

**fill** - fill p; vyplní oblast ohraničenou cestou p

**filldraw** - filldraw p; stejný efekt jako fill p; draw p;

**floor** - floor x; celá část čísla

**for** - for i=1 upto 5: ... endfor; opakování, cyklus for

**forever** - forever: ... exitif i>5; endfor; opakování

**fullcircle** - draw fullcircle; kružnice o jednotkovém průměru

**halfcircle** - draw halfcircle; půlkružnice

**if:elseif:else:fi** - if a<9: ... elseif a=9: ... else: ... fi; podmínka

**input** - input grafy.mf; vložení jiného souboru

**known** - if known a ... fi; výraz je pravdivý, je-li *a* definováno

**left** - z1=left; bod (-1,0)

**length** - length(z2-z1); délka vektoru

**max** - max(a,b,c,d); maximální hodnota množiny

**min** - min(a,b,c,d); minimální hodnota

**mode** - mode=deskjet; nastavení módu

**mode_setup** - mode_setup; nastavení jednotek podle parametrů \mode \mag

**numeric** - numeric a; deklarace proměnné typu číslo

**or** - logický součet

**origin** - z1=origin; bod (0,0)

**pair** - pair b; deklarace bodu

**path** - path p; deklarace proměnné typu cesta

**pen** - pen pn; deklarace proměnné typu pero

**pencircle** - pickup pencircle; pero - jednotková kružnice

**penrazor** - pickup penrazor; pero tvaru jednotkové úsečky

**pensquare** - pickup pensquare; pero - jednotk. čtverec

**pickup** - pickup pencircle scaled 5pt; nastavení pera

**picture** - picture o; proměnná typu obrázek

**point** - point t of p; bod na cestě p v čase t

**quartercircle** - draw quartercircle; čtvrtkružnice

**reflectedabout** - z1 reflectedabout (z1,z2); osová souměrnost podle z1-z2

**reverse** - reverse p; převrácení cesty, body v opačném pořadí

**right** - z1=right; bod (1,0)

**rotated** - z1 rotated fi; otočení kolem počátku o úhel fi

**rotatedaround** - z1 rotatedaround (z2,fi); otočení kolem z2 o úhel fi

**round** - round a; zaokrouhlení

**sind** - sind25; =sin 25ř

**savepen** - p:=savepen; uložení aktuálního pera

**scaled** - z1 scaled 10u; zvětšení, =(x1*10u,y1*10u)

**shifted** - (x,y) shifted (a,b); posunutí, =(x+a,y+b)

**slanted** - z1 slanted a; zešikmení, =(x1+a*y1,y1)

**sqrt** - a:=sqrt(b); odmocnina

**subpath** - subpath (t1,t2) of p; část cesty mezi dvěma časy

**tension** - draw z1..tension a and b..z2; napětí křivky

**transform** - transform t; deklarace proměnné typu transformace

**transformed** - p transformed t; provedení transformace

**undraw** - undraw p; sníží počet překreslení bodů

**unfill** - unfill p; opak k fill

**unfilldraw** - unfildraw p; opak k filldraw

**unitsquare** - draw unitsquare; jednotkový čtverec s levým dolním rohem (0,0)

**unitvector** - unitvector (x,y); jednotkový vektor daného směru

**unknown** - unknown a; true pokud není proměnná definována

**up** - draw z1up..z2; bod (0,1)

**xpart** - xpart b; x-ová složka bodu

**xscaled** - (x,y) xscaled a; zvětšení x-ové souřadnice, =(a*x,y)

**ypart** - ypart b; y-ová složka bodu

**yscaled** - z1 yscaled a; zvětšení y-ové složky, =(x,a*y)

# 4 Tvorba znakové sady programem METAFONT

## 4.1 Co potřebujeme?

Možná to bude znít trochu srandovně, ale co hlavně potřebujeme, je počítač s nějakým operačním systémem. TEX a METAFONT existují snad pro všechny platformy, které jsou dnes používány a dokonce i pro ty, co již nejsou podporovány (jako je TOPS-20, bratranec operačních systémů WAITS, ve kterém byl program TEX vyvinut). Nyní, když je vývoj programů METAFONT a TEX zastaven a váš zdrojový kód bude vždy kompilován se stejnými výsledky na jakékoliv platformě, kde budou programy TEX a METAFONT nainstalovány.

Další potřebnou věcí je nějaký textový editor ukládající pouze „čistý“ text, bez přidaných informací. Takový typ textového editoru bývá implementován normálně na všech platformách (poznámkový blok ve Windows, vi v UNIXu).

Nějakou TEXovou (případně LATEXovou) distribuci, protože TEX nemůže pracovat bez METAFONTu a METAFONT bez TEXu moc nevyužijeme. METAFONT využije s jeho plnou parádou s nějakou TEXovou distribucí a nainstalováním TEXové distribuce se nám automaticky nainstaluje i METAFONT. Ostatní potřebné programy jako GFtoPK a GFtoDVI jsou také součástí distribuce a nainstalují se s ní.

### 4.1.1 UNIXové systémy a jeho klony

Pokud požíváte UNIXový operační systém (Linux, Sun Solaris, BSD atd.) máte velkou šanci, že již máte plnou TEXovou (LATEXovou) distribuci nainstalovanou. Pokud ano, zde se již nic pro vás podstatného nedozvíte a zbytek můžete klidně přeskočit. Pokud nemáte štěstí a žádnou distribuci nainstalovanou nemáte, musíte si ji nainstalovat sami. Jednou z možností je

instalace teTEXu, jedná se o jednu z nejnovějších a lehce instalovatelných distribucí pro UNIXové systémy. Ke stažení je například na adrese www.tug.org/teTEX/. Než se však vydáte na tuto adresu vyzkoušejte napsat v příkazové řádce jednoduchý příkaz „apt-get install tetex-base“. Možná vám to ušetří práci. Například ve Debian GNU/Linux a v dalších „apt-enable“ distribucích. Pokud potřebuje nějaký textový editor ukládající pouze „čistý“ text, tak tomuto požadavku vyhovují editory Vim a Emacs.

### 4.1.2 Windows

V operačních systémech Windows se žádná TEXová distribuce nenachází, takže pokud chcete ve Windows používat TEX a METAFONT musíte ji stáhnout a nainstalovat. Jednou z možností je třeba MiKTEX, který se dá zdarma stáhnout z internetu například na www.miktex.org. Moje zkušenost s touto distribucí (basic-miktex-2.7.2904) však není dobrá. Nepodporuje proof-mode a neobsahuje csplain (pro překlad textů s českými znaky), takže text napsaný v češtině nepřeložíte, dokud si nestáhnete csplain a nedoinstalujete ho. Proč ale věci dělat nadvakrát. Já osobně bych spíše doporučil distribuci TEX Live 2007 (moje verze 1.76a), kterou mi zapůjčil Mgr. Jiří Pech Ph. D., můj vedoucí práce. Tato verze obsahuje vše, co potřebujete a navíc je celá v češtině. Co víc si přát? Jako editor pro kódy stačí bohatě poznámkový blok, který je v každé verzi Windows.

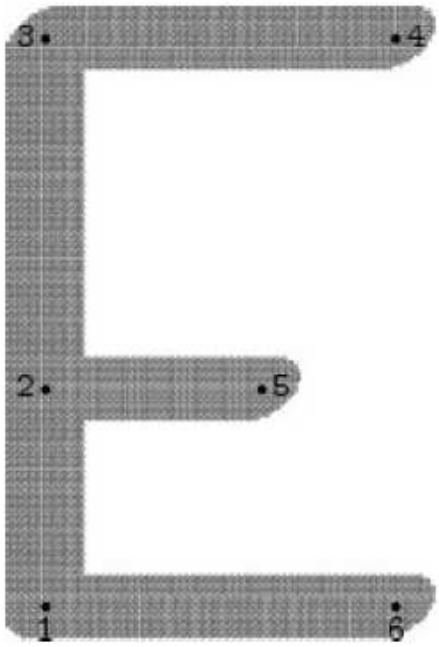
### 4.1.3 Macintosh

Pokud je vaším operačním systémem Mac OS X, pak máte štěstí. Od té doby, co je Mac OS X založený na BSD připomínající zčásti Unix je snadné sehnat velké množství bezplatného softwaru. To zahrnuje i TEX, a teTEX distribuce je k dispozici pro Mac OS X skrz i-Installer nebo fink balíky. Použít

se dá iTEXMac, který se zdá být velmi dobrou teTEXovskou distribucí. Nejsem uživatelem Macu, a nikdy jsem neviděl Mac OS X ve akci, takže nemohu posoudit, jestli jsou dobré recenze, které jsou o iTEXMac, oprávněné, nebo ne.

Na druhou stranu, jestliže je váš operační systém starší verze Mac OS, pak vás pravděpodobně zklamu, ale pro starší verze Mac OS není žádná bezplatná implementace TEXu. Jako dobré implementace se zdají být CMacTEX a OzTEX ale náklady na pořízení jsou u CMacTEX $35, u OzTEX $30, pro jednoho uživatele. Oba dva programy mají místo příkazové řádky příkazové menu, ale stále potřebujete textový editor pro psaní vašich zdrojových kódů. Více informací je k dispozici na http://www.esm.psu.edu/mac-tex/default9.html. Všimněte si také, že tyto distribuce využívají malý slovní trik. Například v implementaci METAFONTu pro OzTEX, nazývané OzMF, vlastní příkaz pro tvorbu fontu není volaný mf ale MakeTEXPK.

## 4.2 První znak




obrázek 11

Písmeno E

Přestože je písmeno A prvním písmenem abecedy pro ukázku prvního znaku v programu METAFONT bude lepší písmeno E. Aby to nebylo až zas tak jednoduché, tak si k tomu něco přidáme: základem našeho písmene bude zlaté číslo, které by nám mělo pomoci k „perfektním“ proporcím našeho písmene. Obvykle se zlaté číslo označuje Φ a jeho hodnota je:

$$\Phi = 1+\sqrt{5}/2 \approx 1.6180339887\ldots$$

A nyní začneme psát. Jsou dvě možnosti jak to udělat. První je řádek po řádce kódu psát do příkazové řádky (spustíme METAFONT příkazem „*mf*“ a pak napíšeme „*\relax*“ ), nebo si můžeme otevřít třeba poznámkový blok a celý kód zapsat do něj. Výběr nechám na vás. V obou případech je výsledek a i postup stejný. První co potřebujeme udělat je nadefinovat si zlaté číslo:

```
Phi = (1+sqrt5)/2;
```

Pozor při odvolání se na proměnnou „*Phi*“, METAFONT je case-sensitive (rozlišuje malá a velká písmena, takže „*Phi*“ a „*phi*“ jsou dvě odlišné proměnné).

Nyní si nadefinujeme všeobecné rozměry našeho znaku, jmenovitě jeho šířku a výšku. Řekněme, že chceme mít znak široký 100 pixelů. To zapíšeme do proměnné „*w*“, takto:

```
w=100;
```

Na začátku jsme si řekli, že náš znak bude založen na zlatém čísle, z kterého budou vycházet proporce znaku, proto si výšku „*h*“ nadefinujeme jako šířku vynásobenou zlatým číslem:

```
h = Phi*w;
```

Jako další si nadefinujeme základní body potřebné k vykreslení znaku. Základní body, které potřebujeme jsou konci čtyř čar, ze kterých se písmeno E skládá (1 vertikální a 3 horizontální). Takže na nakreslení našeho znaku budeme potřebovat 6 bodů. Znak E se skládá ze tří horizontálních čar, každá určená dvěma body a body na stejné čáře musí mít stejné vodorovné souřadnice. Čára mezi body 1 a 6 leží úplně dole, čára mezi body 3 a 4 je na úrovni výšky našeho znaku a čára mezi body 2 a 5 je mezi nimi. Tyto informace zapíšeme takto:

```
y1=y6=0;
y3=y4=h;
y2=y5;
```

Jak můžete vidět, zápis říká nejen o horizontálním zarovnání bodů, ale říká i že body 1 a 6 leží na dně a body 3 a 4 navrchu. Pro body 2 a 5 jsme ale nezapsali přesně jejich y-ovou souřadnici, protože ještě přesně nevíme, kde bude ležet.

Znak E má ještě vertikální čáru, kterou lze zapsat tím, že řekneme, že body 1, 2 a 3 jsou vertikálně zarovnané, vertikální čára je úplně vlevo a i to v našem zápise uvedeme:

```
x1=x2=x3=0;
```

Stále nám zbývá nedefinovat několik souřadnic. Zaprvé, chceme aby dolní a horní horizontální čáry byly stejně dlouhé, jako je šířka znaku. Prostřední čára, ale není tak dlouhá, proto ji musíme udělat o něco menší a k tomu využijeme opět zlaté číslo. Zapíšeme, že délka prostřední čáry vynásobená zlatým číslem je rovna šířce „*w*“:

```
x4=Phi*x5=x6=w;
```

Nyní už zbývá jen doplnit jak vysoko budou body 2 a 5. Musí mít stejnou výšku. Použitím zlatého číslo nadefinujeme výšku bodu 2 tak, aby procento oblasti nad střední čárou a oblasti pod střední čarou bylo rovno zlatému číslu. Jestliže si pamatujete, že substrakce souřadnic dovolí vypočítat vzdálenost mezi dvěmi body, měli byste souhlasit, že dvě oblasti, o něž nám jde, mají mít následující hodnoty: *w*(y3 - y2)* a *w*(y2 - y1)*. Protože se šíře w objela v obou vzorcích, to bude mít za následek její zmizení a my můžeme nakonec vepsat požadovanou rovnost:

```
(y3 - y2) = Phi*(y2 - y1);
```

Všechny informace o pozicích bodů, které METAFONT potřebuje, jsou již zapsané. Nyní si vybereme pero a můžeme začít kreslit. Pero vybereme pomocí příkazu „*pickup*“. Uděláme naše čáry s hladkými konci, proto si vybereme pencircle. První věcí, kterou potřebujeme udělat, je trochu pero zvětšit, zapíšeme:

```
pencircle scaled 15
```

Naše pero chceme takové, aby bylo jako celý znak E založeno na zlatém čísle a kruhového typu. Můžeme snadno splnit obě tyto podmínky, jestliže uděláme pero širší, a podíl jeho šířky a výšky dáme rovný zlatému číslu:

```
xscaled Phi
```

Můžeme dosáhnout hezkého kaligrafického účinku, jestliže naše pero trochu otočíme. Můžeme dělat to použitím příkazu „*rotated*“, ale nejprve potřebujeme definovat úhel. Definice úhlu založeného na zlatém čísle může být udělána mnoha způsoby, ale my chceme něco jednoduchého.V tom případě, stačí nadefinovat jednoduchý vektor. Jednou z jeho souřadnic bude zlaté číslo a druhou bude 1, tím dostaneme úhel na horizontálu pomocí „*angle*“ operátoru.

```
rotated angle (Phi, 1);
```

Celý zápis výběru pera bude vypadat takto:

```
pickup pencircle scaled 15 xscaled Phi rotated angle
(Phi,1);
```

Body jsou určené, pero vybrané a nám zbývá jen znak nakreslit. K tomu použijeme instrukci „*draw*“ a zapíšeme vykreslení čtyř čar, které tvoří znak E:

```
draw z1..z6;

draw z2..z5;

draw z3..z4;

draw z1..z3;
```

První tři instrukce vykreslí horizontální čáry, poslední vykreslí vertikální čáru. Nyní necháme označení našich bodů, aby se objevili na korektuře, kterou se chystáme udělat:

```
labels(range 1 thru 6);
```

Posledním příkazem, který ukončí METAFONT a zobrazí výsledek naší práce bude tento:

```
showit; shipit; end
```

Soubor uložíme například jak ecko.mf. V příkazové řádce najdeme adresář, kam jsme soubor uložili a přeložíme ho:

```
mf  ecko
```

Jako výsledek překladu se nám ve stejném adresáři vytvoří dva soubory: ecko.2602gf a ecko.log. V souboru log jsou uloženy informace o překladu a eventuelně chyby v kódu.

Abychom si mohli náš znak prohlédnout, musíme ještě soubor ecko.2602gf transformovat na soubor DVI. To uděláme opět v příkazové řádce pomocí příkazu:

```
gftodvi ecko.2602gf
```

Nyní nám do adresáře přibyl poslední soubor ecko.dvi. Když si jej zobrazíte v DVI prohlížeči, uvidíte, že vytvořené písmeno vypadá úplně stejně jako na obrázku 11. Celý kód písmene E:

```
Phi=(1+sqrt5)/2;
w=100;
h=Phi*w;


y1=y6=0;
y3=y4=h;
y2=y5;
x1=x2=x3=0;
x4=Phi*x5=x6=w;
y3-y2)=Phi*(y2-y1);
```

```
pickup pencircle scaled 15 xscaled Phi rotated angle
Phi,1);


draw z1..z6;

draw z2..z5;

draw z3..z4;

draw z1..z3;


labels(range 1 thru 6);

showit; shipit; end
```

## 4.3 Runové písmo

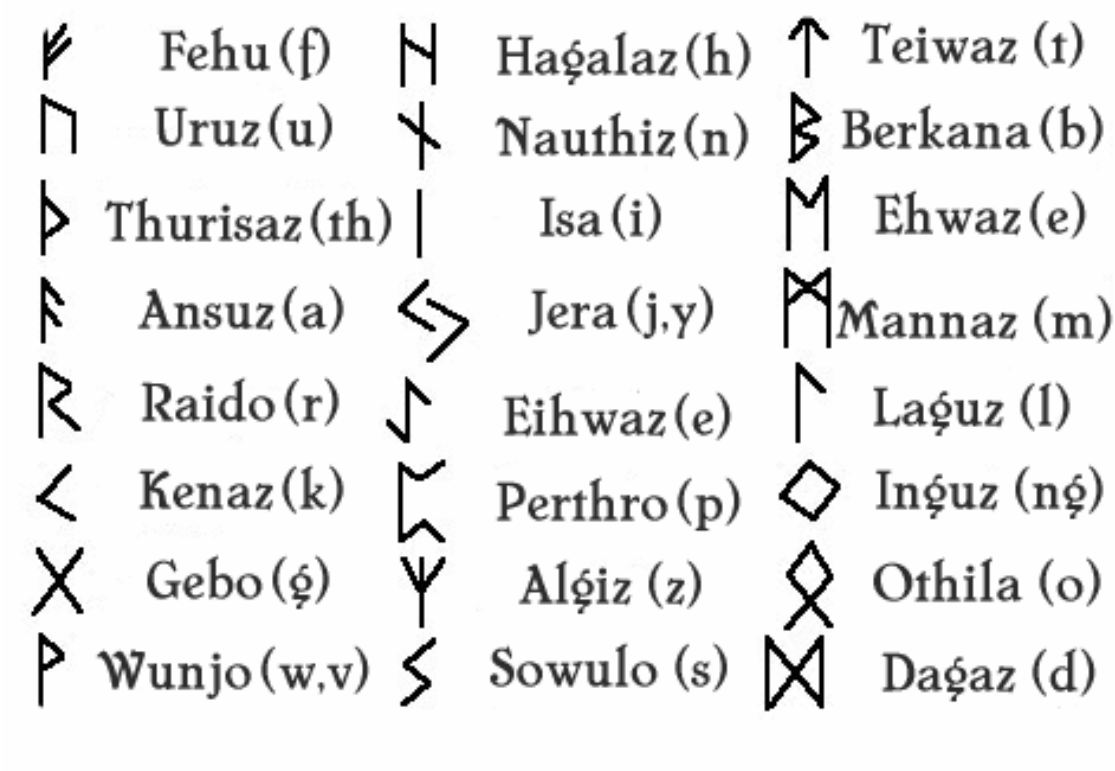
Runové písmo vzniklo asi někdy počátkem našeho letopočtu v podhůří Alp odvozením z mladších podob severoetruských abeced. Některé znaky můžeme z písma Etrusků přímo odvodit, jiné jsou pravděpodobně původního germánského původu. Runové písmo se pak za postupných změn šířilo na sever za Alpy do germánských oblastí a dále do Anglie, Irska a Skandinávie.

Runy se používaly hlavně jako posvátné písmo k zaznamenání důležitých událostí nebo k magické ochraně lidi, nebo věcí. Za prapůvodní runovou abecedu se pokládá 24 run, starší FUTHARK (název odvozen podle prvních šesti run). Runy byly zjeveny Odinovi, když po devět dní a nocí visel na stromě proklát oštěpem v oběť sám sobě.

Existují i další runové abecedy o 16 až 33 znacích, ale protože jako znakovou sadu budu vytvářet starší FUTHARK budu popisovat pouze ten. Ve FUTHARKu dělíme runy do tří skupin (aettir), každá po osmi runách. Názvy

skupin jsou FREY aett, HAGAL aett, TYR aett, názvy jsou odvozeny podle první runy ve skupině.

Runovým písmem se psalo zleva doprava. Písmo se průběžně vyvíjelo, lidé si ho upravovali a slova se dala psát několika fonetickými formami. Proto se můžeme setkat i s jinou podobou run, přidané „nožičky“, zrcadlové zobrazení, atd.




obrázek 12

Starší FUTHARK

## 4.4 Význam run

Význam jednotlivých run:

**FEHU - „F“**

Bohatství je potěšením pro každého, přesto se ho každý musí vzdát ochotně, jestli jásat chce ve Valhalle.

Význam: dobytek-bohatství, peníze, prosperita

**ÚRUZ - „U“**

Divoký tur je prudký, s rohy nad sebou, Nebojácný to bojovník kráčející pustinou, všemohoucí.

Význam: tur-fyzická síla, vytrvalost, odvaha, volnost, mužná síla

**THURISAZ -„Th“**

Thorn je velmi ostrý ke každému, těžko se zdržuje, krutý k těm, kdož odpočívají mezi ostatními.

Význam: Thor/Thurisaz (ledový démon)-nezměrná energie, síla ničit i budovat, zbraň, varování

**ANSUZ - „A“**

Ústa jsou zdrojem řeči, podpora moudrosti a pro každého požehnáním a důvěrou.

Význam: komunikace-moudrost, rozum, věštění, zjevení

**RAIDO - „R“**

Jízda dvoranou příjemná je, silnější však jest na silném oři sedět a urazit míli cest.

Význam: jízda-cesta, kolo, pohyb, poslání, směr

**KÉNAZ - „K“**

Pochodeň pro všechny živé je bledá a jasná, nejvíc hoří tam, kde vznešení

odpočívají.
Význam: oheň-světlo, vidět, vědět, rozumět, osvícení, nápad

**GEBÓ - „G“**

Dar je pro každého sláva a povýšení, a pro potřebné pomoc a potrava.
Význam: dar-svazek, vztah, pouto mezi dárcem a obdarovaným

**WUNJÓ - „V,W“**

Potěšení třeba není těm, kteří malá mají přání a zármutek a rozmnoživši se jsou blažení.
Význam: radost-štěstí, výhra, úspěch

**HAGALAZ - „H“**

Kroupy jsou nejbělejší zrna, vlní se z nebe, točí se ve vzduchu a ve vodu mění se.
Význam: krupobití-náhlá změna, zranění, nemoc

**NAUTHIZ - „N“**

Potřeba v prsou přímá je, leč může často přispěním být, když včas nepečuje se o ni.
Význam: nouze-těžkosti, protivenství, neštěstí, strádání

**ÍSA - „I,Y“**

Led je studený a kluzký, jako sklo třpytí se, jasný je jako drahokamy, pole mrazem slité, jest příjemné zřít.
Význam: led-chlad, zima, kluzkost

**JÉRA - „J“**

Úrodný rok je nadějí pro každého, kdy bohové dovolí zemi dát pestré své plody bohatými a chudým.

Význam: sklizeň-úroda, plodnost země, oslavy, přirozenost

**EIWAZ - „Z“**

Tis navenek jemným stromem je, tvrdým však a pevným v zemi, pastýř ohně s kořeny křivolakými, potěšení na zemi.

Význam: tis-smrt, odchod do Valhally, ochrana před kouzlem

**PERTH - „P“**

Šachy jsouť vždy představení a smích pro hrdé, kde válečníci sedí na hostině radostní pospolu.

Význam: hra-kostky, schopnost ovládat osud, hostina

**ALGIZ - „X“**

Ostřice v kapradí roste, bují ve vodě, pálí krev každého, kdo jí dotkne se.

Význam: sob-úspěch při lovu, ruka odvracející nebezpečí

**SÓWULÓ - „S“**

Slunce je pro mořeplavce spolehlivé vždy, když se pohybují přes lázeň rybářskou, dokud je mořský oř k souši nedonese.

Význam: slunce-vítězství, síla k útoku

**TEIWAZ - „T“**

Tyr jest znamení, které má důvěru vznešených, stále pohybuje se a v temnotě noci nezná odpočinku.

Význam: Tyr-spravedlnost, čest, přísaha, řád, válka, vůle, moc, odpovědnost

**BERKANA - „B“**

Bříza nemá plodů, nese ratolesti bez přírůstků, je nádherná ve svých větvích, obtížena listím, těžká ve větru.

Význam: bříza-obnova, očištění, uzdravení, vyléčení, zrození

**EHWAZ - „E“**

Kůň jest potěšením vznešených, kde reci ve zdraví na ořích svých slova si vyměňují, pro neklidné jest to útěcha.

Význam: oř-pokrok, cesta, důvěra, rychlost, přátelství

**MANNAZ - „M“**

Lidé jsou jejich štěstím a drazí svým příbuzným, přesto se všichni musí odloučit jeden od druhého, neb bohové svěřili tělo zemi.

Význam: člověk-lidství, neodvratnost, úděl, sebeobětování

**LAGUZ - „L“**

Voda se suchozemcům únavnou zdá, jestliže na ni odváží se v člunu nepevném, mořské vlny pěnit budou a mořský oř dbát by hlavou nepohazoval

Význam: voda-proměnlivost, neovladatelnost, nestálost, intuice

**INGUZ - „Ng“**

Ing ponejprv spatřen byl mezi Dány z východu, odplouval přes vlny s vozem svým vzadu, tak válečníci ho pojmenovali.

Význam: naplnění-plodnost, dítě, domov, rodina

**DAGAZ - „D“**

Den poslem bohů jest, světlo bohů pak štěstím a potěchou pro bohaté a chudé.

Význam: den-teplo, světlo, růst

**ÓTHILA - „O“**

Domov je milován každým, když těšit se mohou ze svých práv a práce a prospívat v míru.

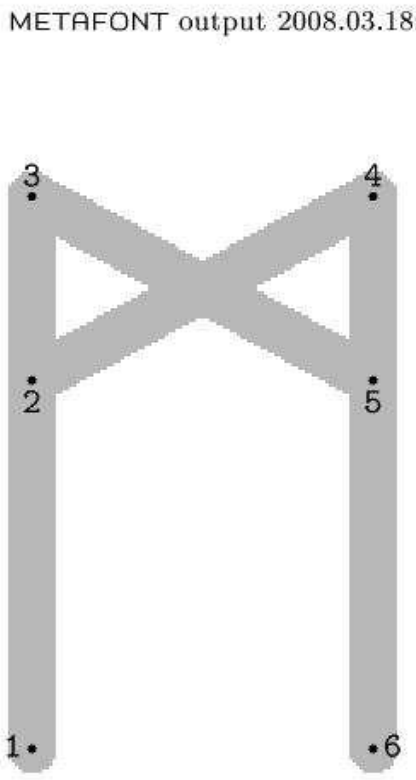
Význam: domov-země předků, odkaz, dědictví, síla rodu

[8]

## 4.5 První runa pomocí METAFONTu

Jako ve všem je nejlepší začít od jednoduchých věcí, a pak přejít ke složitějším. Když se podíváte na obrázek 12, zjistíte, že nejjednodušší runou je runa isa. Tu si ale myslím nemá cenu popisovat, je složená, ze dvou bodů a jedné čáry.

Namísto lehoulinké isa runy, si ukážeme, jak v METAFONTu zapsat runu mannaz. Ta je z hlediska METAFONTového kódu daleko zajímavější, ale přesto je její zapsání stále jednoduché.




obrázek 13

Mannaz runa

K zapsání kódu programu METAFONT si otevřeme nějaký textový editor, ve Windows třeba poznámkový blok, v Linux například KWrite. Při tvorbě runy použijeme naše známe zlaté číslo k určení poměru šířka / výška runy:

```
Phi = (1+sqrt5)/2;
w=100;
h=Phi*w;
```

V prvním řádku definujeme zlaté číslo, ve druhém nastavujeme šířku písmene na 100 pixelů. Třetí řádek nám definuje výšku písmene, která je rovna šířce * zlaté číslo.

Způsobů jak popsat tuto runu v METAFONTu je několik. Jak je vidět na obrázku 13, mannaz runa se skládá ze šesti bodů a čtyř čar. Pokud se na obrázek podíváte důkladně a trochu o něm popřemýšlíte zjistíte, že šest bodů, které tvoří runu jsou body 1, 2 a 3 jsou na stejné horizontále stejně jako body 4, 5 a 6. To lze zapsat třeba takto:

```
x1=x2=x3=0;
x4=x5=x6=w;
```

V našem zápise říkáme, že x-ová souřadnice bodů 1, 2 a 3 je rovna nule (levý okraj) a body 4, 5 a 6 jsou rovny *w* (z anglického weight - šířka), w zde představuje šířku písmene. I když máme popsané x-ové souřadnice pro METAFONT je to málo, musíme popsat ještě y-ové souřadnice bodů.

Z obrázku je vidět, že body 1 a 6, 2 a 5, 3 a 4 leží vždy na stejné vertikále. To můžeme zapsat takto:

```
y1=y6=0;
y2=y5=2/3h;
y3=y4=h;
```

Náš zápis říká, že body 1 a 6 se nacházejí na souřadnici nula (dno písmene), body 2 a 5 se nacházejí ve 2/3 výšky písmene. Výška písmene je určená pomocí proměnné *h* (z anglického height - výška). Jako poslední zbývají body 3 a 4, ty se nacházejí na výšce písmene (*h*).

Nyní máme popsané všechny body, které potřebujeme k tomu, aby se nám vykreslila naše runa. Jediné, co nám zbývá, je popsané body spojit čárou. Pro čáry má, METAFONT, jak víme příkaz „*draw*“, který i použijeme:

```
draw z1..z3;

draw z3..z5;

draw z2..z4;

draw z4..z6;
```

Nyní jsme spojili čarou body 1 a 3, 3 a 5, 2 a 4 a 4 a 6. Aby obrazec vypadal úplně stejně jako na obrázku doplníme kód o poslední dva řádky:

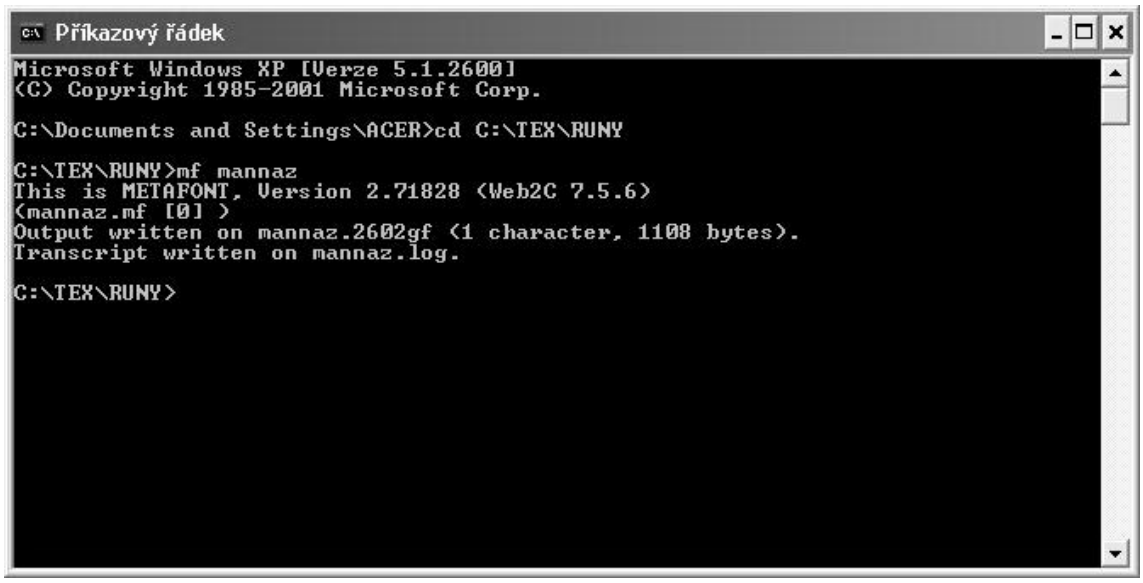
```
labels(range 1 thru 6);

  showit; shipit; end
```

Prvním řádkem říkáme, že chceme vidět nadefinované body v obrazci, druhý řádek se stará o vykreslení našeho znaku. Příkazem end ukončujeme naši definici. Takže kompletní kód mannaz runy vypadá takto:

```
Phi = (1+sqrt5)/2;

w=100;

h=Phi*w;

x1=x2=x3=0;

x4=x5=x6=w;

y1=y6=0;

y2=y5=2/3h;
```

```
y3=y4=h;
draw z1..z3;
draw z3..z5;
draw z2..z4;
draw z4..z6;

labels(range 1 thru 6);
showit; shipit; end
```

Soubor uložíme jako mannaz.mf. Pustíme si příkazovou řádku, přepneme se do složky, kam jsme soubor uložili, a napíšeme příkaz „*mf mannaz*“, proběhne kompilace pomocí programu METAFONT a pokud jsme neudělali nějakou chybu v kódu, bude výsledek překladu vypadat takto:

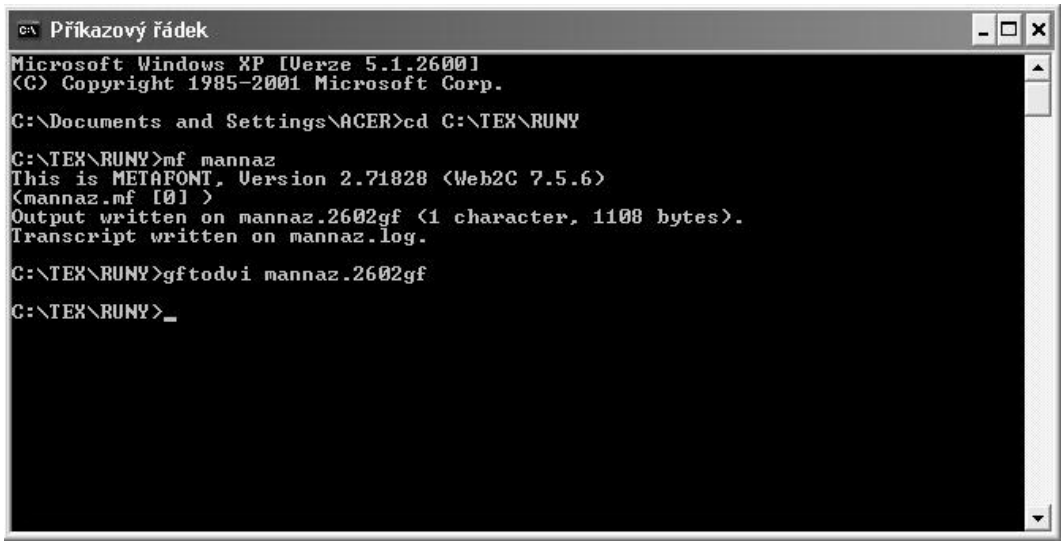



obrázek 14

Překlad kódu znaku METAFONTem

Po překladu nám v naší složce vznikly další dva soubory. Prvním je mannaz.2602gf a druhým je mannaz.log, ve kterém je zapsáno to samé, co vidíme na příkazové řádce. Verze METAFONTu, kterou jsme překládali, a případně chyby, které máme v kódu.

Abychom si mohli prohlédnout naši runu pomocí nějakého DVI prohlížeče musíme napsat ještě jeden příkaz, který nám to umožní. Tím příkazem je: „*gftodvi mannaz.2602gf*“.




obrázek 15

Převod souboru gf na dvi

Nyní se nám vytvořil soubor mannaz.dvi a ten si můžeme již prohlédnout.

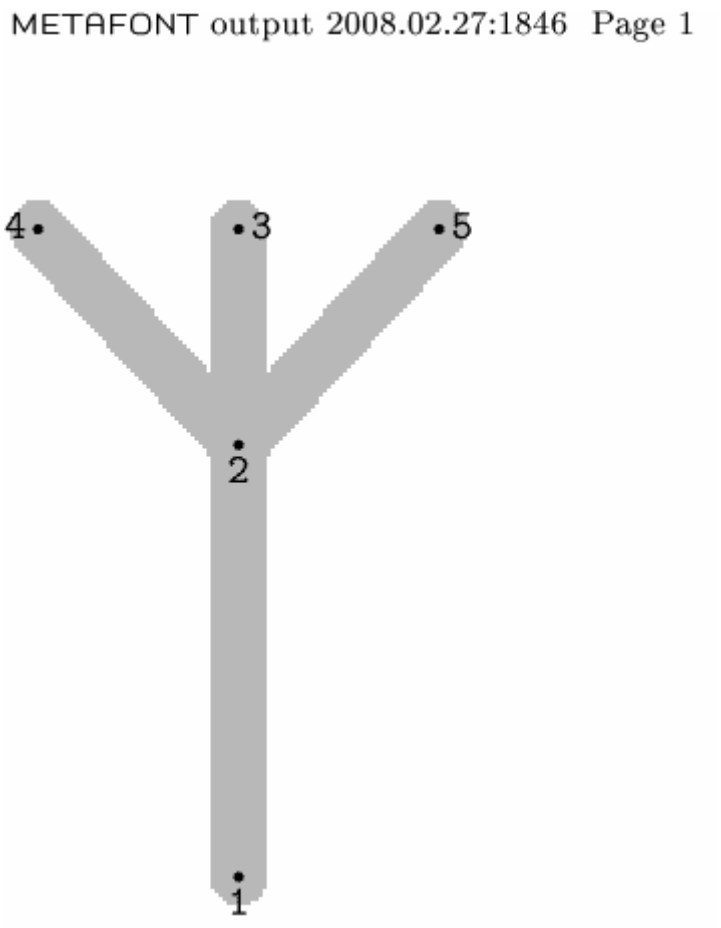
## 4.6 Převedení run do METAFONTového kódu

V minulé podkapitole jsme vytvořili naši první runu. Jedna runa je však málo. Zbývá jich ještě třiadvacet, aby naše runová znaková sada byla kompletní. Psát sem kód každé runy je, myslím, zbytečné. S využitím znalostí z předchozí kapitoly o programu METAFONT není problém dotvořit zbylé.

Někdy to chvíli trvá, ale jak se říká: „Trpělivost růže přináší“. Přesto však ukážu kód několika dalších run, pro inspiraci, dokonce možná i pro vyřešení nějaké nejasnosti. Run, které, si myslím, bylo nejtěžší popsat, a několika dalších run, na kterých ukážu rozdílné možnosti zápisu.

### 4.6.1 Algiz runa

Velmi jednoduchou runou na popis z hlediska METAFONTového kódu je podle mě algiz runa. Skládá se z pěti bodů, které jsou spojeny čtyřmi čarami.

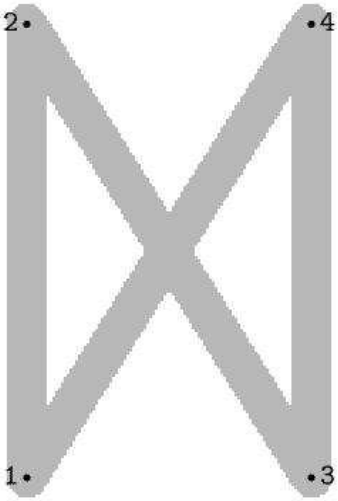



obrázek 16
Algiz runa

Kód potřebný na popsání této runy v programu METAFONT vypadá takto:

```
Phi = (1+sqrt5)/2;
w=100;
h=Phi*w;

x1=x2=x3=w/2;
x4=0;
x5=w;

y4=y3=y5=h;
y2=2/3h;
y1=0;

draw z1..z2;
draw z2..z3;
draw z2..z4;
draw z2..z5;

labels(range 1 thru 5);
showit; shipit; end
```

### 4.6.2 Dagaz runa




obrázek 17

Dagaz runa

Jak je z obrázku runy patrné skládá se ze čtyř bodů a čtyř čar. Nyní ukážu několik způsobů, jak tuto runu zapsat METAFONTovým kódem. Já osobně zvolil pro svůj font způsob popisu pomocí bodů. Jednotlivými příklady chci ukázat, jaké rozdílné možnosti nabízí METAFONT k zapsání stejné věci.

1) popis bodů

```
Phi = (1+sqrt5)/2;
w=100;
h=Phi*w;
z1=(0,0);
z2=(0,h);
z3=(w,0);
```

```
z4=(w,h);
draw z1..z2;
draw z1..z4;
draw z2..z3;
draw z3..z4;
labels(range 1 thru 4);
showit; shipit; end
```

2) popis společných vlastností bodů

```
Phi = (1+sqrt5)/2;
w=100;
h=Phi*w;
x1=x2=0;
x3=x4=w;
y1=y3=0;
y2=y4=h;
draw z1..z2;
draw z1..z4;
draw z2..z3;
draw z3..z4;
labels(range 1 thru 4);
showit; shipit; end
```

3) přes výpočty

```
Phi = (1+sqrt5)/2;
w=100;
h=Phi*w;
x1=x2=0;
x3=x4=x1+w;
y1=y3=0;
y2=y4=y1+h;
draw z1..z2;
draw z1..z4;
draw z2..z3;
draw z3..z4;
labels(range 1 thru 4);
showit; shipit; end
```

4) přes posuny

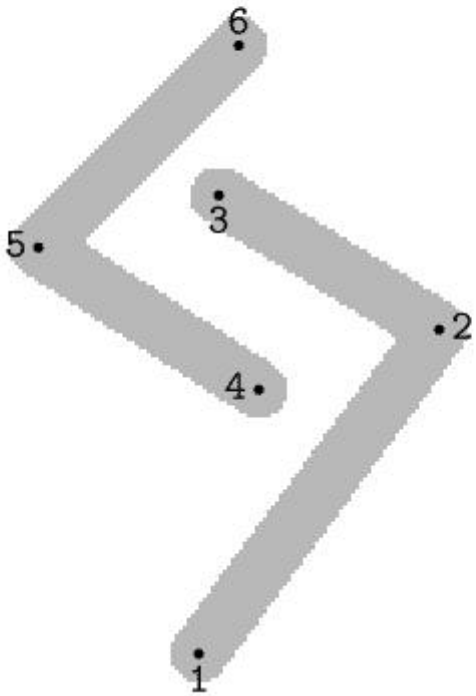
```
Phi = (1+sqrt5)/2;
w=100;
h=Phi*w;
z1=(0,0);
z2=z1 shifted(0,h);
z3=z1 shifted(w,0);
z4=z3 shifted(0,h);
```

```
draw z1..z2;
draw z2..z3;
draw z3..z4;
draw z1..z4;
labels(range 1 thru 4);
showit; shipit; end
```

Ať už runu zapíšeme prvním, nebo kterýmkoliv dalším způsobem, výsledek bude pokaždé stejný. Bude vypadat úplně stejně jako na obrázku 17.

### 4.6.3 Jera runa

Tvorbu této runy jsem si nechal až nakonec. Chtěl jsem si nejdříve na jednodušších runách vyzkoušet různé druhy zápisů. Když jsem si myslel, že problematiku popisování bodů zvládám, vrhl jsem se na jera runu. U tvorby této runy jsem použil taktiku „pokus, omyl“. Runa na první pohled vypadala symetricky, ale symetrická není. Zkoušel jsem popisovat různé body, než jsem dosáhl výsledku, se kterým jsem byl spokojen:




obrázek 18

Jera runa

Zde je kód, který jsem použil pro popsání runy:

```
Phi = (1+sqrt5)/2;

w=100;

h=Phi*w;

x1=w/2-10;

x3=w/2-5;

x2=w;
```

```
y1=10;
y2=h/2+10;
y3=(h-(h/5))-5;
x4=w/2+5;
x6=w/2;
x5=0;
y4=(h/2)-5;
y5=(h-(h/4))-10;
y6=h;
draw z1..z2;
draw z2..z3;
draw z4..z5;
draw z5..z6;
labels(range 1 thru 6);
showit; shipit; end
```

## 4.7 Vytvoření kompletního fontu

Kódy, které byly až doposud uváděny, nám sice vykreslily tvary run, ale pro kompletní kód nejdou použít. Proč jsem se tedy s nimi psal? Důvod je jednoduchý. Sice kódy run, jak byly popsány o kapitolu výše, nyní nepoužiji, ale použiji jejich části. Díky předchozím popisům znám souřadnice bodů, které jsou pro znak důležité. A ty využijeme i v kompletním kódu, jen lehce upravené. Prvním písmenem staršího FUTHARKu je fehu runa a ta je i v ukázce.

Ukázka prvního písmene v kompletním kódu znakové sady run:

```
mode_setup;
u#:=4/9pt#;
define_pixels(u);
Phi=(1+sqrt5)/2;
beginchar(1,10u#,16u#,5u#); "fehu";
     x1=x2=x3=x4=0u;
     x6=x5=w/2;
     y1=0u;
     y5=h;
     y2=h/2;
     y3=h-(h/3);
     y4=h-15;
     y6=y3+10;
     pickup pensquare scaled 15 rotated angle
     (Phi,1);
     draw z1..z4;
     draw z3..z5;
     draw z2..z6;
     labels(range 1 thru 6);
endchar;
```

V kódu se nám objevilo několik nových řádek. Ty popíši později. METAFONT vyžaduje určitou strukturu vstupního souboru. Jako první příkaz se píše „*mode_setup*“, tím přednastavíme některé vnitřní parametry METAFONTu (rozlišení, zvětšení) na hodnoty zadané při jeho spuštění. Dále je dobré na začátku definovat rozměr jednotek, s kterými budeme pracovat.

```
mode_setup;
u#:=4/9pt#;
```

Dalšími novými řádky jsou řádky:

```
beginchar(1,10u#,16u#,5u#); "algiz";
endchar;
```

Jak jste již asi správně uhodli, tyto dva řádky slouží k definici jednotlivých znaků. Kód každé naší runy se musí nacházet mezi těmito dvěma řádky, jinak se nám nepodaří překlad. První řádek určuje začátek definice znaku, jeho číslo ve znakové sadě, rozměry (výška, šířka a hloubka) a jméno znaku. Druhý řádek ukončuje definici znaku.

Teď už stačí stejně jako je zapsaná runa fehu upravit a zapsat do kompletního kódu znakové sady ostatní runy.

## 4.8 Zavedení fontu do TEXu

V této podkapitole si ukážeme, jak náš kompletní kód znakové sady přeložit a kam jej uložit, abychom ho mohli používat v našich dokumentech.

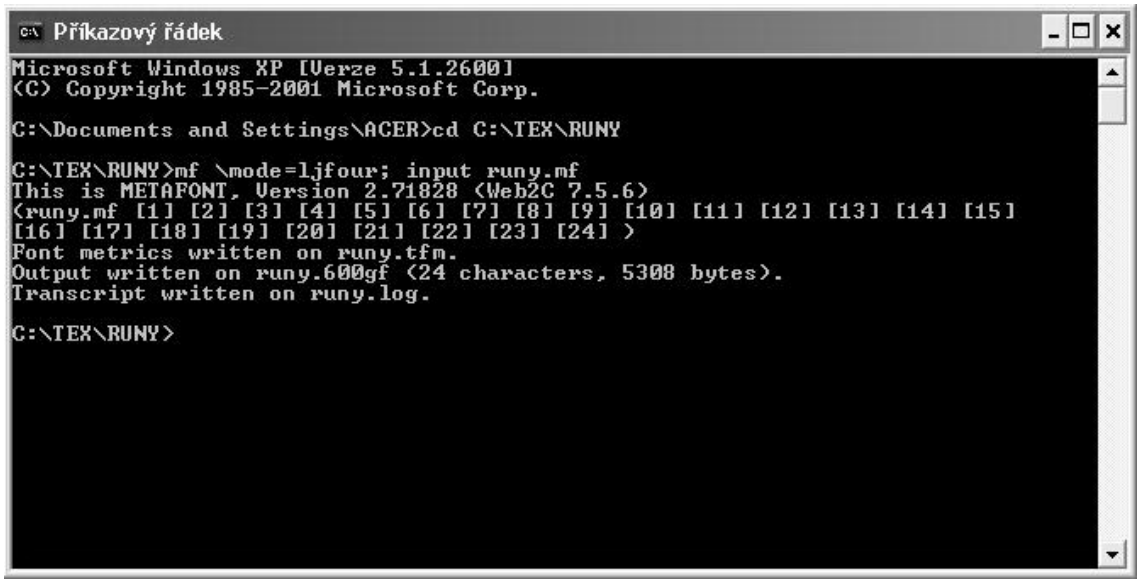
Abychom náš kód převedli na skutečný font, je třeba spustit příkazový řádek a v něm napsat následující příkaz (pochopitelně musíme být v adresáři, kde se nachází kód našeho runového písma):

```
mf  \mode=ljfour; input runy.mf
```

Některé příkazové řádky netolerují obrácené lomítko. To bude mít za následek vypsání chyby. Pokud se vám toto stane, stačí dát argumenty příkazu mf do apostrofů. Upravený příkaz:

```
mf '\mode=ljfour; input runy.mf'
```

Opačné lomítko signalizuje METAFONTu, že musí na příkazové řádce spíše očekávat instrukce než jméno souboru. První instrukce řekla METAFONTu, že má doopravdy pracovat v fontmaking modu (a ne v proofmodu) a poslední instrukce způsobí, že METAFONT nakonec sestaví znakovou sadu run v souboru runy.mf. Výstup příkazu „*mf \mode=ljfour; input runy.mf*“ byl měl vypadat takto:




obrázek 19

Tvorba fontu run

Výstup je lehce odlišný od výstupů, které jsme dostávali při předchozích překladech souborů *mf*. Ukazuje, že tentokrát byly vytvořeny dva soubory. Jestliže zkontrolujete váš adresář, měli byste objevit dva soubory a to runy.tfm soubor a runy.2602gf (nebo s podobným číslem). Nyní je ještě potřeba zabalit gf soubor do použitelného formátu. To se udělá pomocí příkazu:

```
gftopk runy.600gf runy.600pk
```

nebo:

```
gftopk runy.600gf runy.pk
```

záleží na tom, jestli vaše platforma potřebuje neočíslovaný pk soubor. Soubor beta.600pk nebo beta.pk by měl objevit se ve vašem adresáři. A je to! Váš font je nyní dostupný v každém dokumentu, jehož zdrojový kód je ve stejném adresáři jako fontový soubor.

## 4.9 Text v TEXu s využitím run

Jak bylo popsáno již dříve v podkapitole 2.5. Naše runové písmo zavedeme do TEXu pomocí příkazu:

`\font\jméno=`externí jméno fontu `at` požadovaná velikost

Takže v případě našich run to bude:

`\font\runy = runy` (dále by mohlo být uvedeno ještě zvětšení, nebo zmenšení fontu)

Ukázka textu v TEXu s runami:

```
\font\runovePismo = runy

Toto je jera runa \runovePismo jera

\bye
```

## 4.10 Text v LATEXU s využitím run

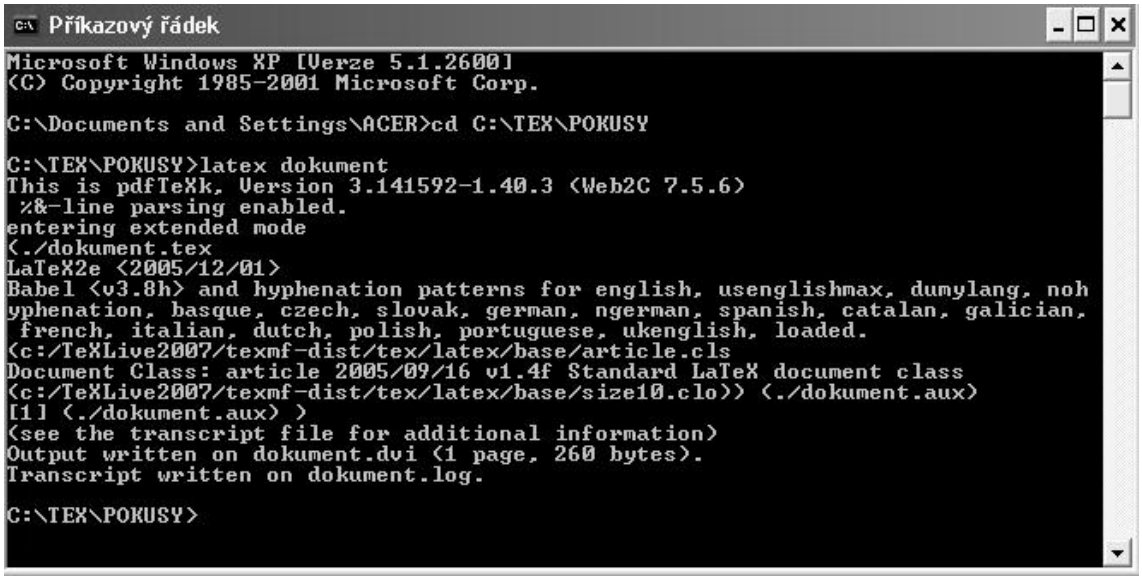
LATEX je balík maker, který umožňuje autorům sázet a tisknout jejich díla v nejvyšší možné typografické kvalitě, přičemž autor používá profesionály předdefinovaných vzhledů dokumentu. LATEX byl původně napsán Leslie Lamportem . LATEX užívá programu TEX jako sázecího stroje.

V této podkapitole si ukážeme, jak naše runové písmo použít v LATEXu. Tato podkapitola je tu z důvodu, že mnoho lidí využívá právě LATEX, který je na syntaxi trochu jednodušší, než TEX.

Ukázka základní kostry dokumentu v LATEXu společně se zavedením runového písma a jeho použití v dokumentu. Řádky uvedené níže stačí zapsat v textovém editoru (poznámkový blok například) a uložit jako dokument.tex.

```
\documentclass{article}
\newfont{\runovePismo}{runy}
\newcommand{\algiz}{{\algizruna}}
\begin{document}
Takto vypadala algiz runa \algiz\.
\end{document}
```

Poté, co máme soubor uložen přes příkazovou řádku, jej přeložíme. Přepneme se do adresáře, do kterého jsme si náš dokument uložili, a v příkazové řádce napíšeme příkaz: *latex dokument.*




obrázek 20

Překlad LATEXového dokumentu

Dokument byl úspěšně přeložen a my si náš výsledek můžeme prohlédnout v dvi prohlížeči.

## 4.11 Problémy se zavedením znakové sady

Jediným problémem, se kterým jsem se během práce na této bakalářské práci setkal, bylo zavedení vytvořené znakové sady do programů TEX a LATEX.

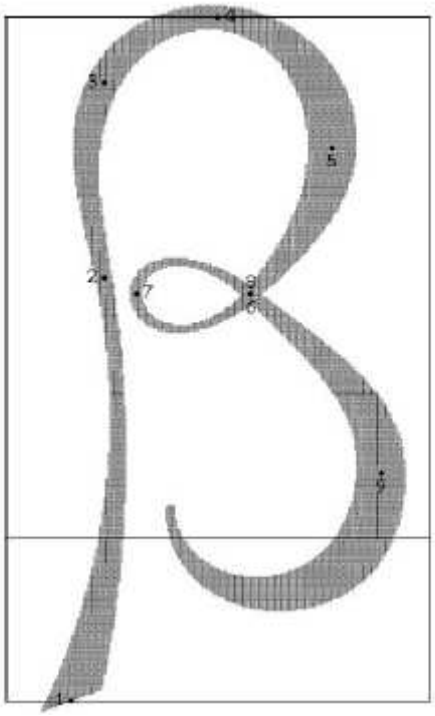
Při zavádění znakové sady run jsem postupoval podle postupů uvedených v kapitolách 4.10 a 4.9, které byly převzaty z použité literatury [1] a [6]. Důvod, proč se mi nepodařilo zavést moji znakovou sadu jsem bohužel neobjevil. Zkoušel jsem podle uvedených postupů zavést jiné znakové sady, jako slabikář pana Olšáka, tím se úspěšně podařilo napsat text v TEXu a řeckou betu z [1], kterou se mi podařilo použít v programu LATEX.

## 4.12 Ukázky složitějších útvarů v METAFONTu

Runové písmo není z hlediska METAFONTového kódu moc zajímavé. Runové písmo se skládá z rovných čar. Ne každého zajímá runové písmo a chtěl by ve svém písmu využít i další věci, jako jsou třeba křivky. Proto jsem se rozhodl přidat ještě tuto podkapitolu, na které chci ukázat ještě další věci, které se pomocí METAFONTu dají vytvořit. Tyto příklady jsou převzaty z použité literatury [1] a [3].

### 4.12.1 Řecká Beta

Znak řecké bety je převzatý z The METAFONTtutorial [1]. Jak vypadá řecké beta, ví asi každý, kdo měl matematiku. Beta vytvořená programem METAFONT může vypadat třeba takto:



obrázek 21

Řecké písmeno Beta

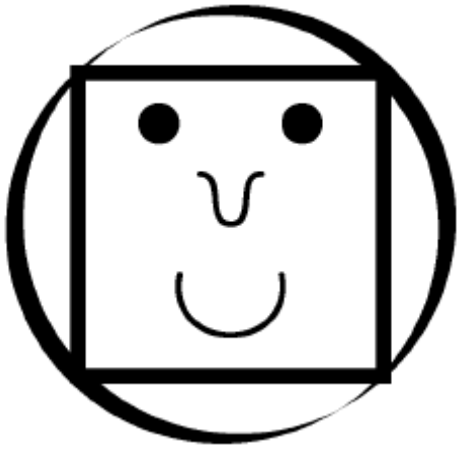
Čára asi v 1/4 obrázku značí výšku řádku. Stejně jako kdybychom psali na linkovaný papír. Co se nachází nad ní, je nad řádkem, co pod ní, to je pod řádkem. Musím se přiznat, že kód tohoto písmene je o dost těžší, než kódy mých run. Posuďte sami jeho „jednoduchost“:

```
u#:=4/9pt#;
define_pixels(u);
beginchar(66,13u#,16u#,5u#); "BETA";
     x1=2u; x2=x3=3u;
     bot y1=-5u; y2=8u; y3=14u;
     x4=6.5u; top y4=h;
     z5=(10u,12u);
     z6=(7.5u, 7.5u); z8=z6;
     z7=(4u, 7.5u);
     z9=(11.5u, 2u);
     z0=(5u,u);
     penpos1(2u, 20);
     penpos2(.5u, 0);
     penpos3(u,-45);
     penpos4(.8u, -90);
     penpos5(1.5u, -180);
```

```
     penpos6(.4u, 150);
     penpos7(.4u, 0);
     penpos8(.4u, 210);
     penpos9(1.5u, -180);
     penpos0(.3u, 20);
     pickup pencircle;
     penstroke
z1e..z2e..z3e..z4e..z5e..z6e..{up}z7e..z8e..z9e..{up}
z0e;
     labels(range 1 thru 9);
endchar;
end
```

### 4.12.2 Trochu jiný smajlík

Po shlédnutí předchozího kódu je třeba se nějak uvolnit. Pomocí programu METAFONT jde kreslit nejen znaky, ale také obrazce. S kreslením obrazců jsme se již setkali v kapitole 3.8, kde jsme vykreslovali obdélník. Myslím, že smajlík je daleko hezčí než obyčejný obdélník, který by každý dokázal nakreslit daleko rychleji pomocí nějakého malovacího programu (i obyčejného malování ve Windows). Kód Smajlíka je převzat z práce Znakové sady v typografických systémech [3] a je upraven, protože původní kód se mi nepodařilo na mém počítači zprovoznit, tak jsem provedl pár malých úprav. Smajlík vypadá takto:



obrázek 22

Smajlík

```
u#:=1mm#;
define_pixels(u);
beginchar(1,20u#,20u#,0);
path p[];
z1=(0,0);
z2=(30u,0);
z3=(30u,30u);
z4=(0,30u);
z5=(10u,10u);
z6=(20u,10u);
z7=(6u,25u);
z8=(10u,25u);
z9=(12u,20u);
z10=(18u,20u);
```

```
z11=(15u,15u);
p1=z1--z2--z3--z4--z1;
pickup pensquare scaled 1.5u;
draw p1;
pickup penrazor scaled 2u rotated 30;
draw z1..z2..z3..z4..z1;
pickup pencircle scaled .6u;
draw z5{dir-100}..z6;
p2=z7..z8..z7..cycle;
fill p2;
fill p2 shifted(14u,0);
draw z9{dir0}..z11..z10{dir0};
endchar;
```

# 5 Závěr

Většina uživatelů TEXu neví skoro nic o programu METAFONT a přitom tyto dva programy spolu tvoří silnou dvojici. Pomocí programu METAFONT můžete vytvořit jakýkoliv znak, nebo obrázek pro váš TEXovský dokument. Nemusíte doufat, že to, co byste rádi použili ve vašem dokumentu, udělal již někdo jiný před vámi, ale udělat si to sami. Odpadne vám i zdlouhavé prohledávání Internetu, které může skončit i nezdarem.

Doufám, že této skupině uživatelů TEXu má práce pomůže přiblížit program METAFONT a pomůže jim zjistit, že to nic zas tak složitého není a začnou jej využívat. Určitě se tato práce nedá srovnávat s encyklopedií o programu METAFONT, The METAFONTBook od Donalda E. Knuta, ale za úvod do této problematiky se považovat rozhodně dá.

# 6 Použitá literatura


1) GRANDSIRE Christophe, The METAFONTtutorial, Version 0.33, 2004, web: metafont.tutorial.free.fr/downloads/mftut.pdf

2) DOOB Michael, Jemný úvod do TEXu, přeložili a upravili Daneš Josef a Veselý Jiří, Praha 1993 web: http://ksp.mff.cuni.cz/study/jemny.pdf

3) ČERNÝ Martin, Znakové sady v typografických systémech, Diplomová práce, Brno 1999

4) OETIKER Tobias, PARTL Hubert, HYNA Irene a SCHLEGL Elisabeth, Ne příliš stručný úvod do systému LATEX2ε, překlad Kočer Michal, Sýkora Pavel, Verze 2.2-Beta, 25 leden 1996, Verze překladu CZ-0.9-Beta, prosinec 1998

5) ŠEDIVÝ Přemysl, BROŽ Miroslav, GŘONDILOVÁ Jana, PÍŠE Michal, HOUFEK Karel, Zpravodaj CSTug 1/98 Kreslíme METAFONTem, ISSN 1211-6661

6) OLŠÁK Petr, První setkání s TEXem, 1999, web: ftp://math.feld.cvut.cz/pub/cstex/docve

7) KNUTH, Donald E., The METAFONTbook. Addison-Wesley, Reading, Massachusetts, 1994

8) http://pohanstvi.net

9) http://valhalla.curaivitkov.cz

10) http://www-kiv.zcu.cz/~herout/html_sbo/metafont/toc.htm

# 7 Seznam obrázků