Ing. Ivan Ryant

VESELÁ RUKOVĚŤ PROGRAMOVÁNÍ

ISBN 80-7062-024-2

OBSAH

1. Proč, pro koho, jak a o čem 7
1.1. Tlačí vás bota? - Choďte bos! 7
1.2. O lidech kolem počítačů 8
1.3. Kudy kam 10
1.4. O světýlkách, pípácích a jiné havěti 13
Slovníček 17
2. Co je třeba vědět 19
2.1. Technické a programové vybavení 19
2.2. Procesor a paměť 20
2.3. Uspořádání paměti 21
2.4. Množiny a typy 23
2.5. Zobrazení dat v počítači 27
2.6. Proměnné 28
2.7. Číselné soustavy 29
3. Zadání úlohy 31
3.1. Co musí tiskárna umět 31
3.2. Nechoďme s kanónem na vrabce 33
3.3. Podrobné zadání úlohy 35
Seznam řídicích znaků a posloupností 43
3.3.1. Styk s obsluhou 43
3.3.2. Styk s počítačem 43
3.3.3. Vnitřní funkce tiskárny 43
3.3.4. Řádkování 45
3.3.5. Stránkování 46
3.3.6. Tisk obrázků 47
3.3.7. Generátory znaků 50
3.3.8. Tabulace 53
3.3.9. Okraje a zpracování textu 55
3.3.10. Písmo 57
3.3.11. Vynechané řídicí posloupnosti 59
3.4. Podrobně o technických prostředcích 60
3.4.1. Pochodem vchod, zastavit stát! 61
3.4.2. Jak se stroje spolu domlouvají 64
3.4.3. Časovač typu 8253 67
3.4.4. Vstupní a výstupní obvod typu 8255 69
3.4.5. Noty 70
4. Rozbor úlohy 73
4.1. Jak na to 73
4.2. Hrubý návrh modulů ZZ a RP 76
4.3. Nekrájejte procesor 82
4.4. Několik slov o záchodové míse 98
4.5. Složité typy dat 101
4.6. Další pán na holení 106
4.7. Kterak půjčovati procesor 109
4.8. Návrh ovladače mechaniky 114
4.9. Odhalení skutečnosti 118

5. Navržené programy 120
Modul ZZ 120
Modul RP 136
Modul GZ 138
Modul GZTAB 146
Modul RVP 152
Modul OU 160
Modul RU 164
Modul RZ 166
Modul OM 168
Modul OP 175
6. O práci ladiče programů 177
7. Na rozloučenou se čtenářem 179

1. Proč, pro koho, jak a o čem

Čtenáři se dostává do rukou příručka, která jej zavede do světa počítačů. Příručka se skládá ze sedmi kapitol. Každá kapitola začíná úvodem, ve kterém je vytčeno téma, následuje několik podkapitol a na závěr je vysvětlená látka stručně shrnuta, aby si čtenář mohl ověřit, že nic důležitého nevynechal. Každá podkapitola začíná obecným úvodem, pokračuje výkladem důležitých pojmů a končí popisem podrobností na příkladu, který je páteří celé příručky.

Obecné úvody kapitol jsou více méně nezávaznou zábavou, která má nezasvěcenému čtenáři přiblížit svět počítačů a má mu ukázat, že i mezi počítači má své místo člověk. Kdo se s tím spokojí, nemusí číst více.

Také při zběžném čtení není nutné zabíhat do podrobností a lze přeskakovat rozsáhlé úseky na koncích kapitol.

Podrobné studium příručky však vyžaduje více než jen poctivé přečtení od začátku do konce. Čtenáři jistě prospěje, když si obtížná místa pečlivě promyslí a třeba i přečte dvakrát. Objeví-li se nejasné pojmy, je třeba vrátit se ke druhé kapitole. Při podrobném studiu 4. kapitoly zase bude nutné jednak se vracet k zadání úlohy (kapitola 3), jednak sledovat vykládanou látku na výsledném řešení úlohy (kapitola 5).

Jedná se tedy o četbu náročnou a hutnou, která klade nároky na čas i na soustředěnost čtenáře.

První kapitola obsahuje tři části. První část má náborově propagační charakter a měla by nejen přilákat ty čtenáře, jimž je toto dílko určeno, nýbrž i odradit ty, jimž určeno není. Druhá část pojednává o nedostatcích v práci s počítači, které přiměly autora, aby vzal pero do ruky a vysvětlil své představy o správné tvorbě programů. Třetí část pojednává o metodě, kterou autor použil.

1.1. Tlačí vás bota? - Choďte bos!

V této podkapitole se budeme zabývat otázkou, komu je určena Veselá rukověť programování.

Autor si dovolil v nadpisu této kapitoly lapidárně vyjádřit filosofii (mimochodem dnes velmi rozšířenou), která vede mnoho lidí k tomu, aby se zřekli práce s počítačem. A skutečně není divu, že řady vyznavačů tohoto světového názoru stále více mohutní. V některých podnicích se nevypłácejí mzdy, protože je zpracovává počítač. Někoho tahá počítač po soudech za to, že odmítá platit částku 0.00 Kčs za elektřinu. Jinému zase počítač

pravidelně zasílá astronomické telefonní účty. V obchodech není zboží - to víte, pane, sklad řídí počítač... Rozvodovost nebezpečně stoupá (od té doby, co se lidé seznamují přes počítač). A to vše se odehrává za halasného vytrubování, že bez počítačů nelze žít a že kdo neumí programovat, není gramotný. Najednou máte pocit, že tady něco nehraje, že lidé slouží počítačům, místo aby počítače sloužily lidem.

Pak máte dvě možnosti: buďto si sbalíte svých pět švestek a začnete poustevničit (a chodit bos) někde na samotě u lesa, anebo se rozhodnete donutit počítače, aby vám sloužily. Zvolíte-li druhou možnost, narazíte na nepřekonatelnou hradbu příruček a učebnic, které jsou psány pro někoho úplně jiného, než jste vy. Nakonec se třeba i naučíte pět základních příkazů jazyka BASIC, jež některým lidem umožňují naprogramovat libovolnou úlohu, ale stejně zjistíte, že pořád nemáte ani páru o tom, jak si ochočit ten ďábelský stroj. Tak si sbalíte svých pět švestek a zbytek svých dnů strávíte někde na samotě u lesa.

Jenže právě teď se karta obrátila. Milý čtenáři, autor ti blahopřeje k tomu, že máš v ruce Veselou rukověť programování. Pokusí se vysvětlit a ukázat na příkladu, jak se má zadávat úloha programátorovi a jak má programátor se zadáním úlohy naložit. Přitom se snaží o to, aby čtenář v každém okamžiku chápal smysl počínání, jež je zde popisováno.

Výklad vychází z pozice zákazníka, který úlohu zadává, a směřuje k postupnému vyřešení úlohy až na úroveň, která vyžaduje podrobnou znalost nějakého programovacího jazyka a zčásti i znalost zařízení, na kterém bude program pracovat. Výklad je určen jak osobám s vážným zájmem o programování, tak těm, kteří zadávají úlohy programátorům. Cílem Veselé rukověti je vysvětlit, co může člověk od počítače chtít a jak to má chtít. Cílem je také ukázat, jak má programátor pracovat se zadáním úlohy.

Není cílem naučit čtenáře nějaký programovací jazyk ani seznámit čtenáře s obsluhou nějakého určitého počítače. Pokud však čtenář zná základy programování v nějakém jazyce nebo základy číslicové techniky, může to být jedině ku prospěchu věci.

Veselá rukověť se dost podrobně zabývá jehličkovými tiskárnami a jehličkovým tiskem. Proto je možné, že upoutá pozornost dalších skupin čtenářů. Může posloužit buď jako příručka pro uživatele jehličkových tiskáren, které zajímá, co v sobě tiskárna ukrývá, nebo jako příspěvek do diskuse konstruktérů tiskáren.

1.2. O lidech kolem počítačů

V této podkapitole se autor pokouší vysvětlit, proč napsal Veselou rukověť programování.

Lidé, kteří pracují s počítači, se dělí do čtyř skupin. První skupinu tvoří lidé, kteří rozhodují o tom, co mají počítače dělat. Tito lidé si vymýšlejí a zadávají úlohy pro počítače. Pro jednoduchost jim říkejme zákazníci.

Druhá skupina lidí určuje, jak se bude úloha řešit. Tito lidé píší programy pro počítače a nazývají se programátoři.

Třetí skupina lidí vytváří technické prostředky, kterými jsou úlohy řešeny. To jsou konstruktéři počítačů.

Čtvrtá skupina jsou ti, kteří počítače obsluhují a používají při tom hotové programy. To jsou uživatelé.

Často se setkáváme s tím, že některého člověka nemůžeme přesně zařadit do žádné z uvedených skupin. Zákazníci se často stávají i uživateli hotových programů, skupina programátorů je zase bohatě rozvrstvena, takže jedni programátoři zadávají úlohy jiným programátorům a navíc všichni programátoři používají při své práci další programy, takže vlastně patří i mezi uživatele.

Tato složitá dělba práce na jedné straně umožňuje efektivní řešení úloh, ale na druhé straně s sebou přináší i řadu nedostatků. Zmíněné skupiny se totiž nedokážou mezi sebou navzájem dobře dohodnout.

Zákazníci nevědí, co všechno mohou od počítače chtít. A na druhé straně často zadávají úlohy, které jsou sice snadné pro člověka, ale velmi obtížně se programují. A to ještě není všechno. I když se stane, že zákazník přijde s úlohou, která je pro počítač jako stvořená, obvykle neumí úlohu zadat. Programátor pak musí ze zákazníka obtížně páčit, co vlastně chce. V horším případě programátor prostě napíše libovolný program a vnutí ho zákazníkovi pod záminkou, že přesně splňuje zadání úlohy. A protože zákazník si ani nedovede představit, jak by mělo řešení vypadat, nechá si třeba naprogramovat výpočet kvadratické rovnice některou z metod umělé inteligence, zakoupí syntaktickou analýzu LALR (15) k počítání výplat anebo přistoupí na to, že před vytištěním abecedního seznamu členů *První jízdní stíhací perutě* musí jména ručně setřídit podle abecedy.

Uživatelé si zase často stěžují na jednotvárnou práci náročnou na pozornost. Nevědí, že mají od programátorů tvrdě vyžadovat jen takové programy, které obsluhu co možná nejméně zaměstnávají, poskytují jí co největší pohodlí při práci a ohlídají chyby. To však také znamená, že by uživatelé měli vědět, jak má takový "**přívětivý**" program vypadat.

Konstruktéři počítačů zase vybavují počítače schopností vykonávat důmyslné instrukce, které se však při programování vůbec nepoužijí. Zato některé zhusta používané a potřebné činnosti je nutné programovat velmi složitě a zdlouhavě. Prostě konstruktéři často vytvářejí jen pomníky svého důmyslu místo strojů, které lze programovat.

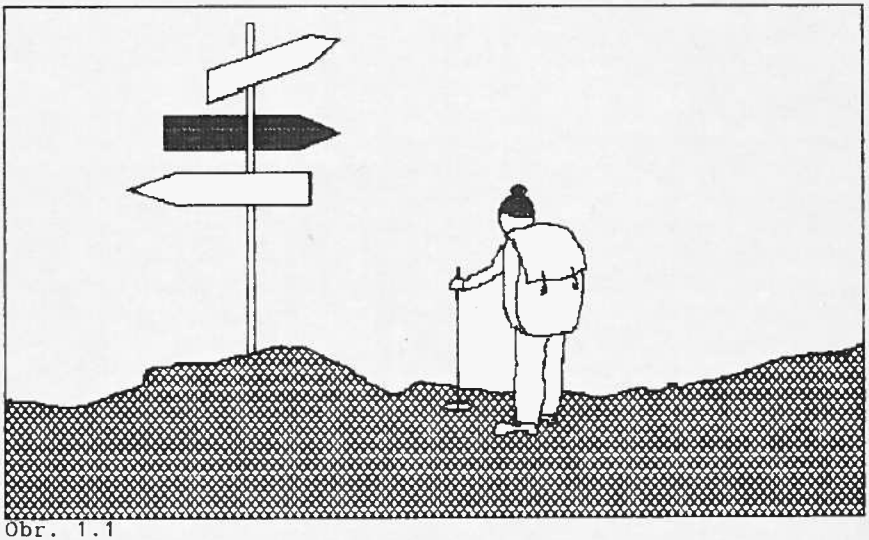
A úplně nejhorší jsou programátoři. Mnozí z nich totiž vůbec neumějí programovat. Vždyť jako kvalifikovaný programátor může

být zaměstnán i vysokoškolák, který absolvuje během svého studia pouhé dva semestry základů programování v jazyce FORTRAN!

Tak co vlastně můžeme od počítačů chtít? Až příliš často jsou tyto mocné nástroje svěřeny do rukou pracovníků, kteří ve vztahu k nim v podstatě postrádají jakoukoli kvalifikaci. Je to něco podobného, jako když dáte dřevorubeckou sekeru do rukou malému dítěti a nutíte je štípat pařezy.

1.3. Kudy kam

V této podkapitole se budeme vènovat výběru metody použité při výkladu vytčené látky.



Obr. 1.1

Jak se lidé učí programovat? Nejčastěji asi tak, že si pořídí (nebo dostanou) nějaký počítač, vezmou si příručku programovacího jazyka BASIC a začnou řešit úlohy typu: napiš "AHOJ" na displej nebo vypočítej 2+2. Po několika měsících pilného učení dokážou tisknout kondiciogramy, naprogramovat hru piškvorky a vypočítat neplodné dny. Získají dojem, že stejným způsobem, jako se dělají několikařádkové programy, lze naprogramovat cokoli - a taky to skutečně dělají: lepí řádek k řádku a plodí tak tisíciřádkové obludy zavšivené desítkami skrytých a záludných chyb, programy, které nelze pozměňovat podle nových požadavků ani přenášet na jiné počítače.

Podstatně lepší metodu volí vysoké školy: napřed učí studenty, jak se programuje, a teprve po zvládnutí metod psaní programů vyučují nějaký programovací jazyk. Studenti se však v počátečním období vůbec nedostanou k počítači, takže si nemohou

prakticky vyzkoušet, co se naučili. Navíc používané školní příklady nemají valného praktického významu a studenti proto dost dobře nechápou, k čemu jim může programování sloužit. Tato metoda je rovněž obvyklá v našich i zahraničních učebnicích programování. Příkladem může být dnes už klasická učebnice od prof. Niklause Wirtha Systematické programování.

(N. Wirth: Systematisches Programmieren,
B.G.Teubner, Stuttgart, 1975;
slovenský překlad: ALFA a SNTL, Bratislava, Praha, 1981)

Se svéráznou a dnes již dobře známou metodou výuky programování přišel Ing. R. Pecinovský, CSc. Zaměřil se na děti a s pomocí populární postavičky robota Karla je učí, jak psát programy. Je to do podrobností propracovaná a v praxi ověřená metoda, která děti dobře motivuje. Hra s Karlem děti baví. Děti učí Karla různým činnostem a tím, že učí Karla, učí se samy tvořit programy. To, co se naučí na Karlovi, potom v pozdějším věku snadno přenesou do prostředí skutečného programovacího jazyka a praktických úloh.

Taková metoda je však pro řadu dospělých nepřijatelná. Dospělý člověk si odmítá "jenom" hrát, byť by taková hra byla sebeužitečnější. Dospělý člověk se chce od samého začátku zabývat "důležitými" problémy, chce ukázat svému okolí, že pracuje a ne že si hraje s neskutečným robotem na pokládání a sbírání značek. Dopřejme tedy dospělým iluzi skutečné úlohy a vážné práce a nabídněme jim řešení "důstojného" problému.

Jednoho pošmourného podzimního dne seslala Štěstěna na autora Veselé rukověti jeden takový dostatečně důstojný problém. Jednalo se o program pro řídicí mikropočítač umístěný v jehličkové tiskárně. Pro ty čtenáře, kteří nechápou, o co jde, vysvětleme jen tolik, že tiskárna je velmi užitečné a důležité zařízení, pomocí kterého počítače zaznamenávají výsledky své práce na papír.

Proč si autor vybral právě tuto úlohu?

Za prvé se jedná o skutečnou úlohu (na rozdíl od školních příkladů odtržených od skutečnosti). Je to úloha menšího rozsahu - jeden programátor ji může zvládnout asi za rok, včetně rozboru zadání. Úloha je dostatečně složitá, takže vyžaduje systematický rozbor a řešení. Mnoho zákazníků a programátorů vůbec netuší, že rozbor a řešení úlohy má svou strategii a taktiku. A ti, kteří to vědí, jen těžko shánějí informace o základech systematického rozboru a řešení úloh.

Za druhé: naše úloha vyžaduje jen velmi málo znalostí o počítačích. Budeme se zabývat holým počítačem s nejjednodušším technickým vybavením. Pro začátečníka je jistě velmi obtížné pochopit, proč a jak se používají třeba magnetické disky. Zařízení, kterými se budeme zabývat my, jsou daleko názornější.

Jedná se o tlačítka, světýlka, elektromagnety nebo motory. A kdo si je nedokáže představit, může si je za pár korun koupit a sám vyzkoušet. Náš řídicí mikropočítač není předem vybaven žádnými programy, takže se vyhneme obtížným pojmům "soubor" a "operační systém".

Volba příkladu vlastně umožnila autorovi, aby sdělil čtenářům co nejvíce poznatků a použil k tomu co nejméně prostředků. S použitím nepatrných základních znalostí se zmocníme tak účinných programovacích nástrojů, jako jsou paralelní procesy, strukturované programování a některé bezpečnostní zásady (umožňující předcházet chybám).

Celkové řešení úlohy, které je ve Veselé rukověti rovněž uvedeno, má jednak orientační význam, jednak má posloužit jako názorná ukázka. Proto není třeba, aby se čtenář snažil pronikat do podrobností.

Když jsme si nyní vyjasnili, **o čem** Veselá rukověť pojednává, zbývá ještě vysvětlit, **jak** je veden výklad. Autor čtenářům jenom vysvětluje, jak program vytvořil (přesněji řečeno, jak by jej byl měl vytvořit, kdyby se při psaní programu vyvaroval všech závažných chyb). Čtenář je vtažen do řešení problému a je na něm, aby se rozhodl, zda chce programování dále studovat do hloubky z nějaké běžné učebnice, nebo mu postačí role zasvěceného zákazníka, který však sám programovat nedokáže.

Přitom je výklad veden dost netradiční metodou **shora dolů**, tedy směrem od zadání úlohy postupným zpřesňováním řešení až na úroveň, kterou lze zvládnout běžnými prostředky programovacích jazyků. Čtenář si tak může být stále jist, že navrhované řešení skutečně řeší zadanou úlohu, a bude vědět, proč se co dělá, k čemu slouží to, co je mu předkládáno. Zůstane mu však do poslední chvíle utajeno, zda doopravdy existují prostředky (nebo zda se dají vytvořit takové prostředky), kterými bychom mohli navrhované řešení uskutečnit. V tomto směru prostě musí věřit autorovi, že takové prostředky lze najít.

Dlužno říci, že takový přístup k okolnímu světu nám není nikterak cizí. Mnoho lidí umí řídit auto, ale málokdo by si je uměl sám postavit. Skoro každý umí obsluhovat televizor, ale málokdo přesně ví, jak uvnitř funguje, proč hraje a ukazuje obraz. Prostě takové složité věci považujeme za černé skříňky, u kterých nás nezajímá, co je ukryto vevnitř. Zajímá nás pouze, jak je máme správně a účinně využívat, tedy jak se chovají navenek, jaké mají vazby s naším světem. Sice se občas stává, že naletíme novinovým zprávám typu "vědci z university v Hlásných Mokropsích dokázali příčinnou souvislost mezi zvýšením vývozu Prazdroje a snížením pohlavní aktivity klokamů rudých", ale v podstatě proti tomu nedokážeme nic dělat. A tak si docela klidně necháváme nalhávat nejrůznější zjevné nesmysly, spoléhajíce se na schopnost svého selského rozumu rozeznat informaci od

dezinformace, a jen v koutku duše se obáváme, že jsme skutečně klamáni tam, kde to nejméně čekáme.

Opakem metody shora dolů je metoda **zdola nahoru**. Zatímco u metody shora dolů si nemůžeme být zcela jisti, zda opravdu používáme jenom ty prostředky a nástroje, které máme skutečně k dispozici (a musíme vystačit s dobrou vírou, zkušeností a intuicí), u metody zdola nahoru naopak ze zadaných jednoduchých prostředků budujeme stále složitější a dokonalejší prostředky vyšší úrovně, o nichž si však nemůžeme být zcela jisti, že jsou skutečně vhodné k řešení zadané úlohy. U nás doma se tomu říká prašť jako uhoď.

Čtenářům, kteří již zvládli základy programování, Veselá rukověť pomůže zařadit jejich poznatky do souvislostí praktických úloh. Autor doufá, že začínající programátoři najdou ve Veselé rukověti důvody pro dodržování některých zásad při programování, které jsou sice v učebnicích hojně doporučovány, nicméně zůstává tajemstvím, k čemu jsou dobré. Těchto zásad se programátoři zbavují coby obtížného břemene hned v počátcích své odborné dráhy, aby jich pak znovu nabývali trnitou cestou poučení z vlastních chyb.

Na závěr této úvodní kapitoly chce autor upozornit, že se snažil o srozumitelný, leč hutný výklad. Proto doporučuje čtenářům, aby četli pozorně a bez chvatu a aby neváhali vracet se k látce, která již byla vyložena, když narazí na obtížné místo v textu. Pokud však výklad zajde do příliš velkých podrobností, které čtenáře prozatím ještě nezajímají, lze přeskakovat celé odstavce až do konce příslušné kapitoly. Autor vyjadřuje tímto svou naději, že způsob i téma výkladu budou natolik zajímavé, aby upoutaly pozornost čtenářské obce.

1.4. O světýlkách, pípácích a jiné havěti

V této podkapitole se budeme věnovat volbě odborných výrazů, které jsou použity ve Veselé rukověti programování.

Potká-li se Eskymák se Zulukafrem, nelze předpokládat, že by se mohli domluvit. Pokud to nejsou programátoři. Všichni lidé na světě, kteří se zabývají počítači, totiž používají společnou řeč, jejíž slova jsou odvozena z anglických termínů. Vzhledem k tomu, že každý normální programátor vystačí s prvním pádem a s třetí osobou jednotného čísla vyjádřenou způsobem oznamovacím v čase přítomném, nevznikají ani mluvnické zádrhely.

Toto pravidlo má ovšem výjimku. Zatímco William Shakespeare se asi obrací v hrobě, když programátoři na celém světě loudují z fajlů do bafrů (a to výhradně v čase přítomném prostém), pak

duše Matěje Kopeckého může klidně odpočívat: jazyk Kašpárkův není a ani nebude prznèn. To je dáno jak tím, že v současné době nemáme okolnímu světu dohromady co říct o počítačích, tak i tím, že tvůrci ČSN 36 9001 se výrazně zasadili o to, aby Čech nedokázal pochopit, co mu říká okolní svèt. Než ubohý čtenář přelouská výraz "technické vybavení počítače" nebo "vyrovnávací paměť", dávno už neví, co se mu o tom "technickém vybavení" nebo "vyrovnávací paměti" říká. Dalibora naučila nouze housti, českého programátora učí česká odborná literatura číst anglické příručky.

V české normě však nenajdeme zdaleka všechny odborné výrazy, které se k nám dostávají z cizokrajných knih a časopisů. Takové nenormalizované výrazy se pak zpravidla používají ve své původní podobě. Potíž je ovšem v tom, že různé firmy mnohdy používají různé výrazy pro týž pojem:

- "setřást" neboli "zdrcnout" lze vyjádřit slovem "compress", "repack" nebo "squeeze";
- "vymazat" se řekne "delete", "erase" nebo "remove";
- "ovladač" je "handler" nebo "driver";
- ukončení určité části programu lze vyjádřit slovy "release", "exit" nebo "leave".

A navíc by mnohý Čech rád budil dojem, že je majitelem, znalcem nebo aspoň uživatelem západní techniky. Takový člověk pak strká ribony do printru a píše na pejpr. Tato podmanivá hantýrka inspiruje redaktory zábavných časopisů ke zprávám o nových kartách do personálních kompjúterů, k úvahám o kompjúterizaci edukačního procesu apod.

Přejímání cizích slov má však jednu nespornou výhodu: jednoznačnost. Zatímco Američan bude považovat "debugger" za prostředek proti obtížnému hmyzu, "pipe" v něm vyvolá představu vodovodu, "stack" bude spojovat se zemědělstvím nebo se sexem a "bootstrap" bude považovat za něco, co už dávno vyšlo z módy, každý Čech okamžitě pochopí, že se jedná o odborné výrazy.

Potíže však začnou, jakmile se má dotyčný Čech rozhodnout, zda bude psát "BASIC" nebo "bejzik", "byte" nebo "bajt" apod. Programátoři zásadně neuznávají jiný pravopis než ten, který znají z anglosaské literatury. Začátečník si však pravděpodobně poplete bit s bytem a obojí bude nejspíš pokládat za apartmá neboli kvartýr, místo aby se vůbec začal pídit po nějakém jiném významu těchto slov.

Strážci čistoty českého jazyka a normalizátoři odborných výrazů prosazují dvě zásady: 1. Všechno překládat do češtiny. 2. Pokud se snad už stalo takové neštěstí, že cizí slovo zlidovělo mezi širokými masami uživatelů jazyka, pak psát českým pravopisem (modul, proces, konektor, generátor) - už kvůli skloňování.

V zájmu spravedlnosti musíme uznat, že někteří z těchto ochránců mateřštiny projevili máchovský cit a velkorysost, když si dovolili vytvořit tak vkusná slova jako "počítač" (computer) nebo střadač (accumulator) a vydolovat z hlubin zapomnění takové

skvosty, jako je "rozhraní" (interface), "uváznutí" (deadlock) nebo "spřažený" (on line). Nic to nestojí a přitom je to účelné i krásné.

Je ke škodě věci, že se až příliš často setkáváme s novodobými čistonosoplenami a klapkobrnkostroji - které nakonec stejně nikdo nepoužívá: elektronický číslicový počítací stroj (místo počítač), zaváděcí program (místo zavaděč - anglicky "loader"), počáteční zavaděč (bootstrap), vyrovnávací paměť (buffer), slabika (byte), technické a programové vybavení počítače (hardware, software). Zdá se být rozumné, aby

slovo bylo tím kratší,
čím častěji se používá.

A jak se s cizími výrazy vyrovnal autor Veselé rukověti? Především se snažil volit jazyk blízký začátečníkům. Domnívá se totiž, že čtenář znalý anglických termínů snadno pochopí, že "bajt" je "byte" a "proces" je "process". Ze stejného důvodu zvolil i zápis programů ve formalizované češtině. Autor předpokládá, že takové programy budou přístupnější pro čtenáře, kteří vůbec neumějí programovat. Konec konců, Veselá rukověť není učebnicí žádného programovacího jazyka.

Ve Veselé rukověti je důsledně dodržováno normalizované názvosloví (ČSN 36 9001) s jedinou výjimkou: není používáno slovo "slabika", nýbrž "bajt".

Potíže mohou čtenáři působit slova "údaj" a "data". Slovem "údaj" je vyjadřován v jednotném čísle týž pojem, který je v množném čísle vyjadřován slovem "data". Slovo "data" je sice běžnější a odvozují se od něj výrazy jako "datová struktura" nebo "typ dat", ale nemá jednotné číslo. Výrazy "struktura údajů" a "typ údajů" (které norma rovněž připouští) se v živé řeči vůbec nepoužívají a jednotnému číslu "údaj" se čeští mluvčí vyhýbají jako čert kříži.

Autor se přidržel (možná ke škodě věci) i normy "Zákonné měřicí jednotky" (ČSN 01 1300) v případě nezákonné měřicí jednotky "palec" a důsledně používá vyjádření ve tvaru "25,4 mm". Dovoluje si tedy alespoň upozornit čtenáře, že tento zákonný, leč nejapný výraz má cosi společného s půlcoulovými trubkami a palcovými titulky.

Mnoho odborných výrazů si autor také vymyslel: dal přednost výrazu "jehličková tiskárna" před "bodová", "maticová" nebo "mozaiková" (tři posledně jmenované výrazy lze použít i pro tiskárny tepelné, laserové a elektrostatické). Použil výrazu "světýlko" místo "svítivá dioda" nebo "LED". Vymyslel si výrazy "pečlivý", "ledabylý", "široký" a "zhuštěný" tisk, "přeškrtnutá" a "neškrtaná" nula, "tisk obrázků" a "pípák". Od učeného kolegy Pecinovského převzal slovo "nadtrhávat".

Jisté rozpaky mu působila anglická (nebo snad latinská?) slova "subscript" a "superscript", která Němci překládají jako "index" a "exponent". Vzhledem k příbuznosti obou pojmů a k tomu, že matematici píší indexy jak dolů, tak i nahoru, rozhodl se autor označit obojí slovem "index" a podle potřeby to případně upřesnit slovem "nahoře" nebo "dole" (až se však čtenář setká s tímto výtvorem v textu, asi se neubrání dojmu, že je to výraz poněkud obludný).

Anglické slovo "pica" autor nepřekládal ani nepřepisoval podle české výslovnosti vzhledem k tomu, že se jedná o cizokrajnou základní typografickou míru, která se u nás vůbec nepoužívá. Vždyť měřicí jednotky jako aršíny, versty, yardy, pudy, gallony, kilogramy a metry se také nepřekládají.

Rovněž slovo "download" není ve Veselé rukověti překládáno. Vzhledem k tomu, že se tento pojem pouze vysvětluje, leč nepoužívá, není pro něj vůbec zaveden český termín, nýbrž vysvětlení (opis) "nahrať znaky v generátoru".

Autor chápe, že čtenáře s rozvinutým citem pro čistotu českého jazyka asi popouzí svévolná tvorba nových slov. Autor proto zakouší značné uspokojení nad skutečností, že pravděpodobně nebude přítomen okamžiku, kdy tyto řádky vyvolají ve čtenáři bouři nevole a snad v něm probudí i agresivní pudy. Nuže, milý čtenáři, co nadělášš? Ať se ti to líbí nebo ne, autor bude své stručné, jednoznačné a výstižné novotvary používat netoliko pro jejich krásu, nýbrž především za účelem stručného a jasného vyjadřování. Autor doufá, že si trpělivý čtenář na nová slova časem přivykne.

Pro úplnost uveďme česko-anglický slovníček odborných výrazů, použitých ve Veselé rukověti. V hranatých závorkách je uveden přibližný přepis výslovnosti těch slov, která se jinak píší a jinak vyslovují. Znalec angličtiny snad promine jistou nedokonalost tohoto přepisu - cílem bylo pouze poskytnout přibližnou informaci těm, kteří angličtinu vůbec neznají.

Slovníček – Vocabulary

adresa	address [adres]
bajt	byte [bajt]
barvicí páska	ribbon [ribn]
bit	bit
data	data [dejta]
élite	élite, elite [ejlít]
fronta	queue [kjú]
generátor znaků	character generator [kerektr dženerejtr]
index	script [skript]
jehličková tiskárna	dot matrix printer [printr]
konektor	connector [konektr]
krokový motor	stepper motor [stepr moutr]
kurzíva	italics [iteliks]
ledabylý	draft
linka	row [rou]
modul	module [modul]
nadtrhávat	overscore [ouvrskór]
národní sada znaků	national character set [nejšjonl kerektr]
nejméně významný bit	LSB, least significant bit [líst signifikant]
nejvýznamnější bit	MSB, most... [moust]
neškrtaná nula	non slashed zero [slešd zíro]
okénko	grid
okraj	margin [márdžin]
paměť	memory, storage [storidž]
pečlivý	NLQ, near letter quality [nýr letr kvolity]
pípák	beeper [bípr], sounder [saundr]
podtrhávat	underline [andrlajn]
port	port
procedura	procedure [prosídžr]
proces	process [proces]
procesor	processor [procesor]
program	program [progrem]
programové vybavení	software [softvér]
proměnná	variable [veriebl]
proporcionální	proportional [proporšjonl]
prostrkání	inter-character spacing [interkerektr spejsing]
předvolba	configuration switches [konfigurejšn svičiz]
přepínač	switch [svič]
přeskok perforace	skip over perforation [ouvr perforejšn]

přeškrtnutá nula	slashed zero [slešd zíro]
přívětivý	user friendly [júzr frendly]
ROM-generátor	ROM-generator [dženerejtr]
rozhraní	interface [interfejs]
rozteč	spacing [spejsing]
ruční	off line [of lajn]
RWM-generátor	RAM-generator [dženerejtr]
řádek	line [lajn]
řádkování, řádkuj	line feed [lajn fíd]
smaž	delete [dylít]
spínač	switch [svič]
spřažený	on line [lajn]
stránkování, stránkuj	form feed [fíd]
tabulace	tab, tabulation [tabulejšn]
tabulátor	tab, tabulator [tabulejtr]
tabuluj	tab, tabulate [tabulejt]
tisk obrázků	bit image printing [bit imidž], graphic printing [grafik]
tiskací hlava	print head assembly [hed asem-]
středění	centring
světýlko	indicator light [indikejtr lajt]
svislá tabulace	vertical tab [vertykl]
široký	expanded
technické vybavení	hardware [hardvér]
tlačítko	push button [puš buton]
tučný	boldface [boldfejs]
typ	type [tajp]
údaj	datum [dejtum]
vodorovná tabulace	horizontal tab [horizontl]
vyrovnávací paměť	buffer [bafr]
vývod	pin
zarážka tabulátoru	tab stop
zarovnání	justification [džastyfikejšn]
zdvojení linek	double strike [dabl strajk]
zdvojení sloupců	emphasizing [emfasajzing]
zhuštěný	condensed [kondensd]
zvyklost IBM / EPSON	IBM/EPSON mode [aj-bí-em, mód]

Původní významy některých anglických slov:

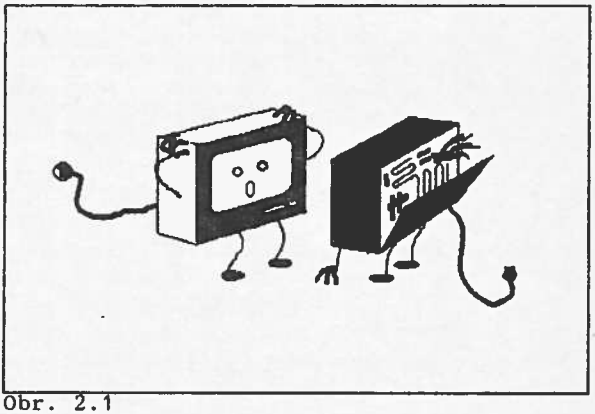
bootstrap	poutko na botě sloužící při obouvání, "štruple"
buffer	nárazník
debugger	odvšivovač
draft	náčrtek
grid	rošt, mříž, souřadnicová síť
pipe	roura
stack	stoh

2. Co je třeba vědět

Autor se snažil počítat s tím, že leckterý čtenář nebude mít žádné předběžné znalosti. Veselá rukověť je určena čtenářům nejrůznějšího vzdělání, povolání i věku. Autor se nechtěl omezit jen na středoškoláky, kteří ovládají základy programování, teorii množin, jakož i přepočty racionálních čísel ze stosedmičkové do třistapětašedesátkové soustavy.

Pro ty, kteří se odvážili budovat své životní štěstí a snad i odbornou kariéru bez tzv. druhé gramotnosti (tj. bez dovednosti dorozumět se s počítačem), autor předkládá několik následujících odstavců. Činí tak v pevné víře, že to (spolu se zdravým rozumem) bude stačit k seznámení čtenářů s řešením úlohy podstatně složitější, než jaké jsou schopny řešit oběti úplného středoškolského vzdělání s maturitou a se znalostí programovacího jazyka BASIC.

2.1. Technické a programové vybavení



Obr. 2.1

Když se podíváme na počítač, zjistíme, že se jedná o menší nebo větší bednu naplněnou spoustou součástek, drátů a desek s plošnými spoji. Tomu se říká technické vybavení nebo také hardvér, anglicky hardware (tj. tvrdé zboží, železářství). Toto technické vybavení je však samo o sobě zhola neužitečné - asi jako hudební nástroj bez hudebníka. Tím hudebníkem je v případě počítače program, tedy předem připravený návod, co a jak má počítač dělat. Programové vybavení počítače se nazývá také někdy slovem českým softvér nebo anglickým software (= měkké zboží).

Co jsou to programy? Jaká je jejich podstata? K čemu slouží? Program není věc, na kterou se dá sáhnout nebo která se dá vidět. Je to představa programátora, jeho nápad, myšlenka, příkaz hmotě, aby se chovala podle přání člověka. Je to něco jako duše vložená do neživého stroje. Mnohému programátoru se při čtení těchto řádek jistě dme hruď pýchou. Dotyčný dost možná zakouší pocit

stvořitele nebo aspoň rabiho Löwa uplácavšího pověstného Golema. Ale prosím Vás, dámy a pánové, jakou metafyzickou cestou byste chtěli vnucovat hmotě své myšlenky? Stejně je nakonec musíte zcela přízemním a často přímo otravným způsobem svěřit papíru, magnetickému disku nebo elektronické paměti. Takže zjišťujeme, že programování (tak jako kdeco na tomto světě) má jak ideální, tak i hmotnou stránku.

Technické vybavení je tvořeno převážně předměty hmotné povahy na rozdíl od programů, které jsou spíš povahy ideální. Patří sem věci, které se pohybují, hřejí, svítí, věci, na které si můžeme sáhnout, podívat se na ně. Jenomže to vše je důmyslně uspořádáno tak, aby to mohlo účelně sloužit programům. Jinými slovy: konstruktér vyjádřil své myšlenky uspořádáním těchto věcí.

Duši jsme přidali trochu tělesna, tělu trochu duševna a programátoři s konstruktéry mohou oprašovat středověké spory duše s tělem, což také náružívě činí, aniž by si (podobně jako ve středověku) kdokoli z nich dělal čáku na vítězství. Jedni bez druhých se prostě neobejdou.

2.2. Procesor a paměť

Počítač řadíme mezi stroje na zpracování informací. Informace zpracovávané počítačem nazýváme

údaje neboli data.

Odbočka ve výkladu (lze při zběžném čtení vypustit): Americký vědec německo-židovsko-maďarského původu Jan von Neumann se svého času geniálně domníval, že ani programy nejsou nic jiného než zvláštní druh dat, že se tedy dají zpracovávat jinými programy (případně mohou zpracovávat samy sebe). Na Harvardské univerzitě tomu dodnes docela neuvěřili (programy tam pro jistotu dávají do jiného druhu paměti než data), přestože Neumann spolu s indiánským náčelníkem Aikenem již koncem čtyřicátých let uskutečnili své myšlenky v počítači, který se stal vzorem všech dalších počítačů na světě. Odbočka v odbočce ve výkladu: Autor se jen domnívá, že Howard Aiken byl indiánským náčelníkem. Dokázat to však nemůže. Konec obou odboček.

Co počítač potřebuje k tomu, aby mohl zpracovávat informace? Musí mít část, ve které uchovává data (ať už data určená ke zpracování, hotové výsledky nebo mezivýsledky - tedy stavy své činnosti). Tato část počítače se nazývá paměť. Při zpracování dat se postupně mění její obsah, paměť tedy přechází ze stavu do stavu, takže postupně zachycuje jednotlivé fáze činnosti počítače, podobně jako okénka na filmovém pásu zachycují pohyb.

Druhá část počítače (ta, která mění obsah paměti) se nazývá procesor. Procesor uskutečňuje činnost předepsanou programem.

Program můžeme považovat za zvláštní druh dat. Program je uložen v paměti v podobě posloupnosti instrukcí. Instrukce je jednoduchý příkaz, kterému procesor rozumí a který umí vyplnit. Procesor čte instrukce jednu po druhé z paměti a okamžitě vykonává to, co mu přikazují. Některá instrukce mu například sdělí, kde má v paměti hledat určitý údaj a zároveň mu přikáže, aby tento údaj přečetl z paměti. Pak procesor přečte další instrukci a ta mu třeba přikáže přičíst k onomu údaji číslo 3. A další instrukce mu uloží, aby výsledný součet uložil do paměti. Takže při vykonávání programu procesor čte z paměti data, provádí s nimi jednoduché operace a zapisuje výsledky do paměti.

Jak ale vložíme data do počítače a jak získáme výsledky z počítače? K tomu slouží zvláštní místa v paměti, která si nepamatují to, co do nich procesor ukládá. Některá z nich slouží ke vstupu dat, jiná k výstupu dat a souhrnně se nazývají

porty (port znamená brána).

Do vstupního portu můžeme vložit data například pomocí klávesnice a procesor si je pak může přečíst. Pokud by procesor chtěl do vstupního portu zapisovat, neprovede se nic a zapisovaná data se ztratí. Do výstupního portu naopak procesor může zapisovat, ale zapsaná data už později nemůže přečíst (přečetl by nesmyslný údaj). Data zapsaná procesorem do výstupního portu se objeví např. na obrazovce displeje nebo uvedou do pohybu typovou páku psacího stroje.

Tento výklad je pochopitelně příliš stručný, než aby mohl být úplný. Autorovi šlo jen o vytvoření hrubé představy v hlavách čtenářů.

2.3. Uspořádání paměti

Nejmenší prvek paměti je takový, který si dokáže zapamatovat 1 bit.

Bit [čti bit]

znamená anglicky kousíček a je to vtipná zkratka pro "binary digit" [bajnery dydžit] = dvojková číslice. Bit je informace, která umožňuje rozlišit dvě hodnoty. Tyto hodnoty mohou být různého

typu,

to znamená, že mohou mít různý (předem dohodnutý) význam: 1 a 0, pravda a nepravda, zapnuto a vypnuto, být či nebýt prvkem množiny apod.

Jednobitovou informaci můžeme snadno zaznamenat pomocí nejrůznějších fyzikálních principů (můžeme udělat nebo neudělat

díru do papíru, zmagnetovat nebo nezmagnetovat citlivou vrstvu magnetické pásky, rozsvítit nebo zhasnout, vydávat vysoký nebo hluboký tón, zapnout nebo vypnout elektrický proud, připojit nebo odpojit napětí). V dalším výkladu se ještě setkáme s tímto způsobem přenosu jednobitové informace: Na jeden konec drátu (neboli vodiče) připojíme napětí. Napětí se šíří drátem skoro tak rychle, jako světlo ve vesmíru, takže vbrzku dostihne druhého konce drátu. Když na konci drátu shledáme napětí menší než 0,8 voltu, budeme si myslet, že začátek drátu je v odpojeném stavu a že se přenáší informace "0", "nepravda", "vypnuto", "není prvkem množiny" apod. Když na konci drátu shledáme napětí větší než 2 volty, budeme si myslet, že začátek drátu je v připojeném stavu a že se přenáší informace "1", "pravda", "zapnuto", "je prvkem množiny" apod. Napětí mezi 0,8 a 2 V nesděluje žádnou informaci a není jasné, co to udělá.

Když chceme uložit do paměti počítače třeba písmeno abecedy (která má desítky prvků), nevystačíme s dvouhodnotovou pamětí. Musíme si pomoci podobně, jako když z písmen vytváříme slova. Písmen je také malý počet, ale slov je nepřeberné množství. Proto je zvykem seskupovat jednobitové paměti do osmic. Taková osmice se nazývá slabika neboli

bajt (často se používá i anglická podoba "byte").

Prvky v osmici jsou navzájem nezámìnné (podobně jako písmena ve slově), takže vytvářejí možnost velkého množství kombinací. V jednom bajtu může být uložena jedna z $2^8 = 256$ různých hodnot. Hodnoty bajtů můžeme znázorňovat třeba tak, že jednotlivé bity v bajtu označíme číslicemi 0 a 1. Jeden bajt může mít tyto hodnoty:

```
00000000
00000001
00000010
00000011
00000100
00000101
00000110
00000111
00001000
   ...
11111110
11111111
```

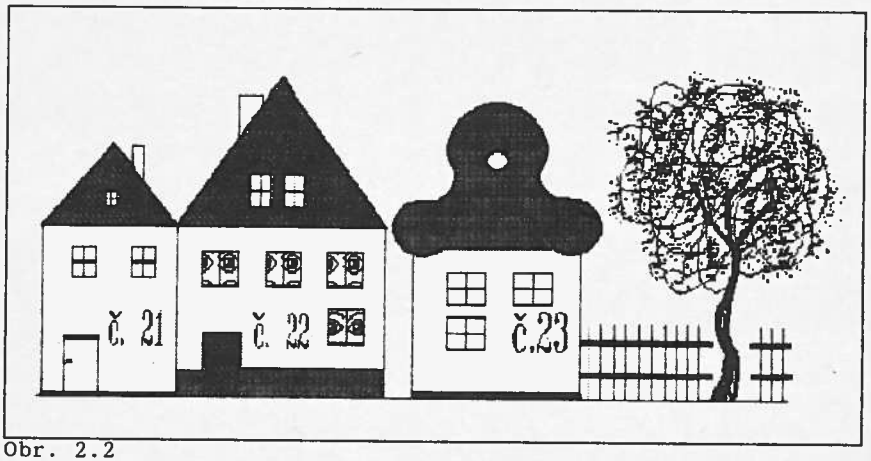
Cvičení: Autor snažně prosí laskavého čtenáře, aby v zájmu snazšího pochopení další látky překonal svou lenost a povšiml si pravidelného opakování nul a jedniček na jednotlivých pozicích v bajtu. Nechť se pak pokusí doplnit ještě aspoň dalších deset hodnot za hodnotou 00001000.

Procesor pracuje s celými bajty najednou, proto je i paměť uspořádána po bajtech. Jednobajtové paměťové buňky jsou uspořádá-

ny a očíslovány podobně jako domy v ulici. Proto se také pořadové číslo paměťové buňky nazývá

adresa.

Když procesor pracuje s pamětí, musí používat adresy bajtů, které chce z paměti číst nebo do paměti ukládat. Adresy najde procesor v instrukcích, které vykonává.



Obr. 2.2

A teď si představme, že i adresy potřebujeme uchovávat v paměti. Jednobajtové paměťové místo nám umožňuje, abychom v něm uchovávali jednu z 256 různých hodnot, které můžeme konec konců považovat třeba i za adresy. Jenže 256 různých adres, to je příliš málo i pro nenáročné úlohy. Proto se adresy vyjadřují jako dvoubajtové hodnoty. Takových hodnot je 256 krát 256, tj. 65536, a to pro malý počítač obvykle stačí. Větší počítače ovšem používají delší adresy (4 bajty).

2.4. Množiny a typy

Pro naše potřeby stačí, když si pojem množina vysvětlíme pomocí jiného slova stejného významu: skupina. A my se pro potřeby dalšího výkladu můžeme omezit jen na takové množiny, které nemohou mít víc než osm prvků (to souvisí s počtem bitů v bajtu a později se k tomu ještě vrátíme). Množina je něco jako anglický klub, který má osm členů. Nikdo jiný nemá do klubu přístup, cizí návštěvy v klubu se netrpí a ani nelze přijmout devátého člena. V klubu je osm křesel (pro každého člena jedno). Do klubu však nemusí přijít každý večer všichni členové, někteří mohou chybět. Jejich židle pak zůstanou volné. Množina přítomných

členů klubu vždy obsahuje jenom ty prvky - členy, jejichž židle jsou právě obsazeny.

V předchozí kapitole jsme se zmínili o hodnotách různých typů, např. o hodnotách nepravda a pravda, které jsou logického typu. To zapíšeme:

```
logický = (nepravda, pravda);
```

Podobně bychom mohli popsat i typ, jehož hodnotami jsou členové našeho klubu:

```
členové = (Andy, Ben, Cyrus, Dick, Ed, Frank, Greg, Harry);
```

Jiným typem hodnot jsou třeba celá čísla, dalším typem znaky abecedy. Popišme nyní typ čísel, která můžeme zobrazit v jednobajtové paměti. Jedná se o interval celočíselného typu počínajíc hodnotou 0 a končíc hodnotou 255:

```
bajt = 0..255;
```

Také množiny mohou být různého typu, jejich hodnoty by však bylo obtížné vyjmenovávat. A také by to nemělo velký smysl. Jde o to, že každý typ množiny je vždy vybudován nad nějakým jiným typem (nad typem prvků množiny), např.:

```
klub = množina nad členové;
Množina_Čísel = množina nad 0..7;
```

Určitá množina typu klub může mít např. tuto hodnotu:

```
[Andy, Ben, Greg]
```

- to znamená, že v klubu jsou právě přítomni jen tři členové - Andy, Ben a Greg. Autor doporučuje pozornosti čtenářů, že hodnoty množin zapisujeme pomocí hranatých závorek.

Vidíme, že typem jsou určeny všechny **hodnoty**, které jsou v rámci typu přípustné. To ale není všechno. Typem jsou určeny i všechny **operace**, které lze s hodnotami provádět. Ukažme si to na typech, které již známe:

Hodnoty *celočíselného* typu můžeme sečítat, odčítat, násobit, dělit a zjišťovat velikost zbytku po dělení. Výsledné hodnoty jsou opět celočíselného typu. Tyto operace zapisujeme:

sčítání	-	3 + 4, 1 + 1 apod.
odčítání	-	3 - 4, 1 - 1, 5 - 2 apod.
násobení	-	3 * 4, 1 * 1, 5 * 2 apod.
dělení	-	3 děl 4, 1 děl 1, 5 děl 2 apod.
zbytek po dělení	-	3 mod 4, 1 mod 1, 5 mod 2 apod.

Vysvětleme, že "mod" je zkratka slova modulo, kterým se označuje operace zjištění zbytku po dělení. Tedy 5 mod 2 je zbytek po dělení čísla pět dvěma = 1, 1 mod 1 = 0, 3 mod 4 = 3. Operace děl je celočíselné dělení, takže 7 děl 2 = 3, 3 děl 4 = 0, 1 děl 1 = 1.

Kromě zmíněných operací lze s celými čísly provádět i takové operace, jejichž výsledkem je logická hodnota:

je_větší, např. 3 > 4 (výsledkem je hodnota nepravda),
je_rovno, např. 1 = 1 (je pravda),
je_menší, např. 2 < 5 (je pravda).
je_menší_nebo_rovno, např. 3 <= 4 (je pravda),
je_nerovno, např. 1 <> 1 (je nepravda),
je_větší_nebo_rovno, např. 2 >= 5 (je nepravda).

Také s *logickými* hodnotami lze provádět operace. Jsou to tyto operace:

a_zároveň

Výsledkem operace a_zároveň je hodnota "pravda" jenom tehdy, když jsou oba operandy pravdivé, např.:

(3 > 4) **a_zároveň** (1 = 1)
je nepravda **a_zároveň** pravda
je nepravda.

Takže

pravda **a_zároveň** pravda je pravda
pravda **a_zároveň** nepravda je nepravda
nepravda **a_zároveň** pravda je nepravda
nepravda **a_zároveň** nepravda je nepravda

nebo

Výsledkem operace nebo je hodnota "pravda" vždy, když je aspoň jeden operand pravdivý, např.:

(3 > 4) **nebo** (1 = 1)
je nepravda **nebo** pravda
je pravda.

Takže

pravda **nebo** pravda je pravda
pravda **nebo** nepravda je pravda
nepravda **nebo** pravda je pravda
nepravda **nebo** nepravda je nepravda

neplatí

Výsledkem operace **neplatí** je opak hodnoty operandu, např.:

neplatí (1 = 1)
je **neplatí** pravda
je nepravda.

Takže

neplatí pravda je nepravda

neplatí nepravda je pravda

Také s *množinami* lze provádět několik operací. Musíme však dbát na to, aby oba operandy byly stejného typu. Jsou to tyto operace:

Průnik, např.:

[Andy, Cyrus] * [Harry, Cyrus] = [Cyrus].

Výsledná množina obsahuje ty prvky, které jsou obsaženy v obou operandech.

Sjednocení, např.:

[Andy, Cyrus] + [Harry, Cyrus] = [Andy, Cyrus, Harry].

Výsledná množina obsahuje ty prvky, které jsou obsaženy aspoň v jednom z operandů.

Rozdíl, např.:

[Andy, Cyrus] - [Harry, Cyrus] = [Andy].

Výsledná množina obsahuje ty prvky, které jsou obsaženy v levém operandu a přitom nejsou obsaženy v pravém operandu.

Operace [Andy, Frank] + [0, 3] je nesmyslná, protože operandy jsou různého typu.

Množiny lze také porovnávat. Výsledkem porovnání je logická hodnota:

[Andy, Frank] = [Andy, Frank] je pravda,
[Greg, Dick] <> [Greg, Dick] je nepravda,
[Greg, Ben] = [Greg, Dick] je nepravda.

Porovnání <= (čteme: je podmnožinou) je pravdivé právě tehdy, když všechny prvky levého operandu jsou obsaženy také v pravém operandu (pravý operand však může obsahovat ještě nějaké prvky navíc), např.

[Andy, Frank] <= [Harry, Frank, Andy] je pravda,
[Andy, Frank, Harry] <= [Harry, Frank, Ben] je nepravda.

Porovnání >= je pravdivé právě tehdy, když pravý operand je podmnožinou levého operandu, např.:

[Andy, Frank] >= [Harry, Frank, Andy] je nepravda,
[Andy, Harry, Frank] >= [Harry, Frank, Andy] je pravda.

Pozn.: Při porovnání množin nelze používat znaménko < ani znaménko >. Pro tato znaménka neexistují žádné operace s množinami.

Také lze zjistit, zda prvek je obsažen v množině, např.:

Andy je_prvkem [Andy, Frank] je pravda,
Andy je_prvkem [Ben, Harry] je nepravda.

2.5. Zobrazení dat v počítači

Čtenář se již možná tváří nedůvěřivě, neboť se mu jistě zdá, že mu autor věší bulíky na nos. Copak celá ta pestrá paleta nejrůznějších typů může být ukládána do šedivé, jednotvárné paměti počítače ve formě bajtů? Jakým lišáckým způsobem bys toho chtěl, bláhový spisovateli, dosáhnout?

Autor tedy již spěchá, aby vše náležitě vysvětlil.

Zobrazit množinu v podobě bajtu je velmi snadné. Předpokládáme, že množina nemůže obsahovat více než 8 prvků (jinak bychom ji museli zobrazit do několika bajtů). Jednotlivé bity v bajtu prostě přiřadíme všem přípustným prvkům množiny podobně, jako jsou členům klubu přiřazena křesla (viz kapitolu 2.4). Jestliže je prvek v množině přítomen, bude příslušný bit obsahovat hodnotu "je_prvkem" (podobně, jako když je křeslo v klubu obsazené). Není-li prvek v množině přítomen, má příslušný bit hodnotu "není_prvkem" (jako když je křeslo prázdné).

Nyní ještě zbývá otázka, jak přiřadíme jednotlivé bity možným prvkům množiny. Řekli jsme si, že pořadí bitů v bajtu je významné. Když totiž bity v bajtu navzájem přeházíme, bude mít bajt jinou hodnotu. Hodnota 00000001 je jiná než hodnota 00000010. Je to podobné, jako když přeházíme písmena ve slově (lízat - zalít, klání - lkání, plesk - sklep, šeří - řeší - šíře - říše).

Bity v bajtu jsou tedy seřazeny. Proto musíme seřadit i prvky, které se mohou vyskytovat v množině. Nejjednodušší je seřadit je ve stejném pořadí, v jakém jsou uvedeny v definici typu. Proto nepravda předchází hodnotu pravda (zapíšeme: nepravda < pravda),

Andy < Ben < Cyrus < Dick < Ed < Frank < Greg < Harry,

0 < 1 < 2 < 3 < 4 < 5 < 6 < 7.

Takže např. hodnoty typu množina nad 0..7 se zobrazí těmito hodnotami bajtu:

[]	se zobrazí hodnotou bajtu 00000000
[0]	se zobrazí hodnotou bajtu 00000001
[1]	se zobrazí hodnotou bajtu 00000010
[2]	se zobrazí hodnotou bajtu 00000100
[3]	se zobrazí hodnotou bajtu 00001000
[7]	se zobrazí hodnotou bajtu 10000000
[1, 2]	se zobrazí hodnotou bajtu 00000110
[0, 2, 3, 7]	se zobrazí hodnotou bajtu 10001101

Než přistoupíme k dalšímu výkladu, je třeba, aby čtenář dobře chápal rozdíl mezi typem prvků množiny a typem množiny. Typ množiny je jiný typ než typ prvků. Typ množiny je vybudován nad typem jejích prvků.

Kromě množin prvků musíme zobrazovat i prvky samotné. Mohli bychom to udělat třeba tak, že bychom je zobrazovali jako jednoprvkové množiny (např. Ben jako [Ben], 3 jako [3]). V takovém případě bychom však nikdy nemohli vměstnat do bajtu všech 256 různých hodnot, které v něm lze zobrazit.

Musíme tedy využít toho, že prvky v množině můžeme kombinovat. Je to podobné, jako když se označují lístky v tramvaji nebo v autobusu: strojek vyznačí do 9 okének na lístku nějakou kombinaci dírek. Okének je sice jenom devět, ale kombinací dírek je 511 (s neoznačeným lístkem 512). Představme si bajt jako množinu bitů:

```
Typ
  bajt = množina nad bit_v_bajtu;
```

Ale jak to udělat, abychom žádnou kombinaci nevynechali? To si nejsnáze ukážeme na zobrazení čísel 0 až 255. Každý bit v bajtu bude představovat nějakou dílčí hodnotu, bude mít určitou váhu. Nejlépe nám vyhoví tento typ:

```
bit_v_bajtu = (v1, v2, v4, v8, v16, v32, v64, v128),
```

kde hodnoty jednotlivých vah jsou zřejmé ze jmen prvků. Poznamenejme, že bit s váhou 128 se nazývá nejvýznamnější bit v bajtu a bit s váhou 1 se nazývá nejméně významný.

Pro každé číslo, které chceme v bajtu zobrazit, budou některé váhy v množině přítomné a jiné nikoli. Součtem vah pak dostaneme zobrazenou hodnotu, např.:

```
[] zobrazuje hodnotu 0,
[v1] ... 1
[v2] ... 2
[v1, v2] ... 1 + 2 = 3
[v2, v8] ... 2 + 8 = 10
[v1, v4, v8, v64, v128] ... 1 + 4 + 8 + 64 + 128 = 205.
```

Poznamenejme, že prvky každého typu jsou uspořádané, takže libovolný prvek můžeme vyjádřit jeho pořadovým číslem (počínajíce nulou).

Nyní bude pro čtenáře jistě hračkou vymyslet, jak budeme zobrazovat hodnoty typu logické = (nepravda, pravda). Množina [] bude představovat hodnotu nepravda, množina [v1] hodnotu pravda.

2.6. Proměnné

Zbývá zodpovědět ještě jednu otázku: jak budeme ukládat hodnoty do paměti, abychom nepopletli jejich typy? Vždyť procesor nepozná, jaký typ má hodnota uložená třeba na adrese 1234. Procesor nám klidně udělá třeba množinový průnik logické hodnoty pravda s číslem 107. Ale to je přece pořádný nesmysl! Proto

budeme hodnoty ukládat do míst předem určených k ukládání hodnot příslušného typu. Takovým místům v paměti budeme říkat

proměnné.

To proto, že hodnota obsažená v proménných se může měnit. Navíc tím, že pro proměnnou vyhradíme předem místo v paměti (říkáme, že proměnnou deklarujeme), můžeme vymezit správnou délku tohoto místa. Hodnotám některých typů totiž nestačí jen jeden bajt, například k uložení adresy potřebujeme dva bajty.

2.7. Číselné soustavy

Bystrý čtenář již možná vidí podobnost našeho kódování čísel se zápisem v desítkové soustavě. Jednotlivé váhy odpovídají řádům, jako jsou jednotky, desítky, stovky atd. Rozdíl je jen v tom, že my máme k dispozici pouze dvě hodnoty: (není_prvkem_množiny, je_prvkem_množiny). Proto se v našem případě jedná o řády jako jednotky, dvojky, čtyřky, osmičky atd., jako když půlíme jablko: napřed na půlky, pak půlky na čtvrtky, čtvrtky na osminy atd. A jaké číslice budeme umísťovat do pozic jednotlivých řádů? Odpověď je nabíledni. Vždyť máme jenom dvě hodnoty, které můžeme označit číslicemi 0 a 1. Protože má naše číselná soustava pouze dvě číslice, nazývá se dvojková. Vzpomeňme si na to, že slovo bit je vlastně zkratkou pro dvojkovou číslici.

Důvtipného čtenáře teď asi napadá: vždyť jsme odjakživa zapisovali hodnoty bajtů pomocí jedniček a nul. Tak to jsou tedy ta slavná dvojková čísla? Baže jsou, odpovídá autor a zde přináší drobný příklad:

Číslo 205 jsme zobrazili v kapitole 2.5 množinou [v1, v4, v8, v64, v128], tedy bajtem 11001101.

Na závěr se ještě zmiňme o jiné číselné soustavě, ke které mají všichni programátoři (nejedná se tedy o patlaly v jazyce BASIC) silný citový vztah. Je to číselná soustava opravdových mužů, kterou někteří programátoři používají, aby dali najevo svou vysokou profesionální úroveň, jiní tak vyjadřují své opovržení lidmi, kteří tuto soustavu neovládají. Jen málo programátorů je k jejímu užívání vedeno důvodem (kterým se však ohánějí i všichni ostatní), a to je přímo směšně snadný převod do dvojkové soustavy a nazpátek. Mnohý čtenář možná již podlehl panice, když si představil dlouhé řádky jedniček a nul, kterými by se podle odstavce o dvojkové soustavě měly hemžit programy. Říká se, že nejhorší smrt je smrt z vyplašení. Tak se, milí čtenáři, neplašte a nemalujte čerta na zeď. Vždyť autor již chvátá s řešením tohoto závažného problému.

Jedná se o soustavu šestnáctkovou neboli hexadecimální. Jak již název napovídá, má šestnáctková soustava šestnáct číslic: 0,

1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Číslice A až F mají hodnoty 10 až 15. Abychom rozeznali šestnáctková čísla od desítkových, budeme je ukončovat malým písmenem h. Hodnotu dvojciferného šestnáctkového čísla vypočteme tak, že hodnotu první číslice vynásobíme šestnácti a přičteme druhou číslici, např.:

3Eh = 3 * 16 + 14 = 48 + 14 = 62.

Pro úplnost dodejme (kdyby si snad někdo ze čtenářů předčítal nahlas), že číslo 3Eh čteme: "tři é šestnáctkově" nebo zkráceně "tři é hexa". A nyní ten snadný převod mezi dvojkovou a šestnáctkovou soustavou. Mnohému možná napoví zajímavý vztah, že totiž

$2^4 = 16$.

Prostě k zakódování jedné šestnáctkové číslice potřebujeme právě čtyři bity a naopak k vyjádření libovolné kombinace čtyř dvojkových číslic potřebujeme právě jednu šestnáctkovou. Jeden bajt tedy vyjádříme pouhými dvěma šestnáctkovými číslicemi a v tom už se přece musí každý vyznat. Když pak budeme používat kódovou tabulku sady znaků, ve které je každému znaku přiřazen kód od 0 do 255, nakreslíme ji v podobě přehledné čtvercové tabulky, jejíž sloupce budou označeny první šestnáctkovou číslicí a řádky druhou šestnáctkovou číslicí kódu. Schválně si tuto tabulku všichni prohlédněte (tab. 3.1). Snadno z ní zjistíte, že kód písmene 'R' je 52h, tedy dvojkově 0101 0010. S převodem na desítkové číslo 82 by méně zdatný počtář mohl mít trochu potíže a při velkém počtu převodů hrozí nebezpečí chyb.

Shrnutí:

Byly vysvětleny pojmy: technické vybavení, programové vybavení, paměť, procesor, data, port, typ, adresa, bit, bajt, množina, proměnná a pojem deklarovat proměnnou. Byly zavedeny typy logický, bajt a množinové typy. Byly zavedeny operace pro uvedené typy. Bylo vysvětleno zobrazování hodnot všelijakých typů pomocí bajtů. Byla vysvětlena dvojková a šestnáctková číselná soustava.

3. Zadání úlohy

Počítače mají mnoho různých druhů vstupních a výstupních zařízení, která jim umožňují získávat informace z okolního světa a předávat výsledky své práce uživatelům. Mezi základní druh výstupního zařízení patří tiskárny. Jsou to přístroje, které píšou na papír nejrůznější texty. Takovými texty mohou být texty programů, účty za telefon a za elektřinu, ale i výzkumné zprávy, obchodní dopisy a pozvánky. Také Veselá rukověť programování byla nejprve vytištěna tiskárnou počítače a potom rozmnožena.

Jako tiskárna může počítači posloužit i elektrický psací stroj nebo dálnopis s rychlostí 10 až 15 znaků za sekundu, ale také velké elektromechanické tiskárny schopné vytisknout několik řádků za sekundu, moderní laserové, elektrostatické a xerografické tiskárny a malé a laciné inkoustové a jehličkové tiskárny. A právě taková malá jehličková tiskárna je předmětem našeho zájmu.

Naše tiskárna bude mít uvnitř jednoduchý počítač. Nejedná se však o počítač, který používá tiskárnu k výstupu svých výsledků. Jedná se o počítač, který bude pouze řídit činnost elektrické a mechanické části tiskárny. Dalším úkolem tohoto malého řídicího počítače bude přijímat znaky z nadřízeného počítače, jehož výstupní údaje bude naše tiskárna psát. Naším úkolem je napsat program pro řídicí počítač tiskárny.

3.1. Co musí tiskárna umět

Tato kapitola je určena čtenářům, kteří dosud nepoužívali žádnou tiskárnu.

Každého asi napadne, že tiskárna musí především umět psát písmena, číslice a všelijaké znaky a značky. Je zvykem, že tiskárna přijímá znak po znaku z počítače a nějakým způsobem převádí každý znak do viditelné podoby na papír. Má-li tiskárna vytisknout slovo 'RUKOVĚŤ', počítač do ní musí poslat znaky 'R', 'U', 'K', 'O', 'V', 'Ě', 'Ť' pěkně jeden za druhým.

Bystrý čtenář však již jistě tuší zradu: co když nechceme tisknout znaky pěkně jeden za druhým? Co když chceme vytisknout tabulku do sloupců pod sebe? Co když chceme začít psát na další řádek? Co když chceme psát na další stránku?

K tomu nám poslouží zvláštní znaky, které tiskárna nepřevádí do viditelné podoby takovým způsobem jako třeba písmena. Takové zvláštní znaky způsobí nějakou zvláštní činnost tiskárny: řádkování, stránkování, umístění následujícího znaku na začátek

řádky nebo do následujícího sloupce tabulky. Protože tyto znaky řídí činnost tiskárny, říkáme jim

řídicí znaky.

Řídicí znaky bývají pojmenovány anglickými zkratkami, např. ESC ze slova escape [čti iskejp] = únik. Vzhledem k tomu, že tyto zkratky mají větší cenu historickou než nápovědnou, nebudeme se zabývat jejich českým významem. Jazykovědná poznámka: anglické zkratky označující řídicí znaky jsou pozůstatkem dávných dob, kdy zvonky ještě zvonily (BELL), tisklo se typovými pákami, typové páky se vozily na vozíku a vozík se vracel do levé krajní polohy řídicím znakem CR (carriage return).

Nároční uživatelé však vyžadují od své tiskárny i takové činnosti, které nelze vyjádřit jediným řídicím znakem. Jsou například takové národy, které nepovažují značku dolaru za vhodné označení peněžní jednotky a vyžadují od své tiskárny, aby místo dolarů tiskla třeba libry nebo jeny. Jsou lidé, kteří místo matematických značek chtějí tisknout panáčky nebo srdíčka. A navíc žijí mezi námi i takoví uživatelé tiskáren, kteří chtějí, aby jejich tiskárna kreslila grafy, schémata nebo obrázky.

Proto musí tiskárna rozumět i celým řídicím posloupnostem znaků, kde význam každého znaku v posloupnosti závisí na tom, jaké znaky mu předcházely. Např. znak '1' má v posloupnosti ESC '1' jiný význam než v posloupnosti ESC 'U' '1'.

K nejsložitějším řídicím posloupnostem patří ty, které slouží k tisku obrázků. Tiskárna považuje papír za množinu bílých míst a chce vědět, která bílá místa má začernit. Řídicí posloupnost pak musí kromě znaků obsahovat i množinu černých míst na papíru.

Na tomto místě již asi ubohý čtenář bije hlavou o nějaký tvrdý předmět a křičí: Aj, zadrž, šílený spisovateli, a přestaň chrlit své nestřídmé nároky na obyčejnou výstupní periférii, neb můj dosud jasný rozum začíná pozbývati své jiskrnosti a v hlavě se mi tvoří temný galimatyjáš. Nemámť ani zbla představy o tom, jak tohle všechno v té tiskárně uskutečníme. Autorovi na tomto místě nezbývá než přislíbit, že v kapitole 3.3 všechny řídicí znaky i posloupnosti ještě podrobně vysvětlí a tím ukojí i náročného čtenáře.

Prozatím jen doufá, že si čtenář vytvořil přibližnou představu o tom, co má tiskárna umět, a dovoluje si pozvat čtenáře na malou přehlídku jednoduchých technických prostředků, jimiž si troufá splnit nastíněné úkoly.

Shrnutí:

Tiskárna musí umět tisknout písmena, číslice a různé znaky a značky. Dále musí umět kreslit obrázky. Činnost tiskárny je řízena řídicími znaky a řídicími posloupnostmi.

3.2. Nechoďme s kanónem na vrabce

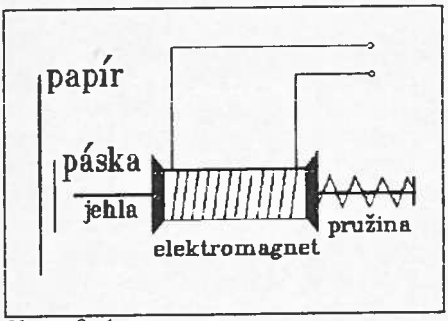
Tato kapitola je určena těm programátorům a konstruktérům mechanických částí tiskárny, kteří neznají podstatu jehličkového tisku.

Ještě asi před patnácti lety byla mechanická část jehličkové tiskárny velkolepým pomníkem svého konstruktéra. Od těch dob se mnohé změnilo - zvětšila se užitná hodnota a klesly náklady na výrobu jehličkových tiskáren. Největší zásluhu na zvětšení užitné hodnoty má to, že tiskárny jsou vybavovány malými řídicími počítači. A největší zásluhu na zlevnění tiskáren má především zjednodušení mechanické části. Dnes se o výrobu tiskáren celkem úspěšně pokoušejí dětské zájmové kroužky i někteří amatéři ve svých domácích dílnách a soukromých kovárnách. Co tedy od mechaniky chceme?

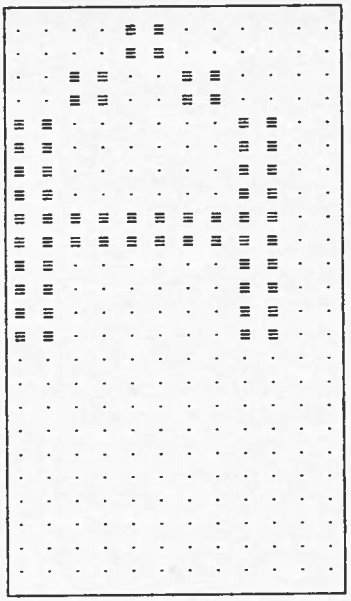
Zjednodušeně řečeno, úkolem tiskárny je očerňovat čistý papír. K tomu slouží barvicí páska, kterou známe z psacího stroje - je to mašle nasáklá černou barvou. Chceme-li využít celou barvicí pásku, musíme ji posouvat - např. převíjet z jedné cívky na druhou. (Modernější jehličkové tiskárny vytahují barvicí pásku otvorem ze zvláštní krabičky a druhým otvorem na opačné straně krabičky nacpávají použitou pásku dovnitř.) Pro nás je důležité, abychom uměli pásku rozjet a zastavit.

Dále potřebujeme něco, čím bychom čas od času přitiskli barvicí pásku k papíru a udělali tak černou skvrnu. K tomu použijeme něco jako ocelový drát - tiskací jehlu (viz obr. 3.1). Jehla musí být tenká, aby černá skvrna byla co nejmenší a tisk dostatečně jemný a přesný. Jehla však nesmí být špičatá, aby nepropichovala barvicí pásku. Jehla je vymršťována elektromagnetem proti papíru. Proti síle elektromagnetu působí pružina, která vysunutou jehlu zasune zpátky. My budeme zapínat a vypínat proud do elektromagnetu. Zapnutím a následovným vypnutím proudu uděláme tečku na papír.

Potřebujeme však tisknout na libovolné místo na papíru. Tiskací jehla je součástí tiskací hlavy, která se může pohybovat nad papírem ve směru řádek pomocí krokového motoru (O krokových motorech se podrobněji zmíníme v kapitole 3.4). Tiskárna obsahuje ještě jeden krokový motor, který posouvá papír nahoru a dolů. Náš program může způsobit posun hlavy o krok doleva nebo doprava a posun papíru o krok nahoru nebo dolů.



Obr. 3.1



Obr. 3.2

A jak budeme z černých teček vytvářet znaky? Každý znak vepíšeme do okénka rozděleného na 24 krát 12 políček (obr. 3.2). Okénko je tvořeno 24 linkami složenými z 12 políček na šířku. Chceme-li tisknout znak po znaku, měli bychom vytisknout vždy celý sloupec 24 políček najednou (pomocí hlavy obsahující 24 jehel). Pak bychom měli posunout hlavu nad další sloupec a to celé opakovat dvanáctkrát.

Skutečně je zvykem vytvořit tiskací hlavu z několika jehel sestavených do sloupečku. (Existují však i jednojehličkové tiskárny, např. československá GAMA - CENTRUM nebo japonská SEIKOSHA.) Ale není technicky možné sestavit všech 24 jehel nad sebe, protože kraje černých teček na papíru se musí překrývat. Proto tiskací hlavy mívají jen 7 až 12 jehel nad sebou (drahé tiskací hlavy ovšem mohou mít až tři takové sloupce jehel vedle sebe - to však není náš případ). Při rychlém a ledabylém tisku se písmena vytvářejí jenom ob linku sedmi až devíti jehlami. Při pečlivém tisku se pak každé písmeno tiskne nadvakrát: napřed ledabyle, pak se posune papír tak, aby se jehly nastavily mezi právě vytištěné linky, a dalším přejezdem hlavy se doplní chybějící linky. Proto je nejlepší, když si tiskárna celý řádek pamatuje, aby se mohl celý řádek napřed vytisknout ledabyle a pak podle potřeby doplnit při druhém přejezdu hlavy. Tiskací hlava naší tiskárny má 9 jehel a my tak máme možnost vytisknout libovolnou kombinaci 9 teček v každé poloze hlavy a papíru.

Naše tiskárna ještě obsahuje malý reproduktor, který nám umožňuje vyluzovat tóny a tím zpravovat obsluhu tiskárny o různých zajímavých událostech (např. když dojde papír a není na co tisknout). Toto zařízení budeme nadále krátce označovat slovem pípák. Pípák podrobně probereme v kapitole 3.4.

Tiskárna dokáže vyhovět nejrůznějším požadavkům uživatele, dokáže se přizpůsobit rozměrům papíru, nabízí několik různých sad znaků atd. To vše lze volit řídicími znaky. Ale má to jednu vadu: jakmile se tiskárna vypne, všechno zapomene a po zapnutí se uvede vždy do stejného výchozího stavu. Pokud však uživatel potřebuje zásadně českou abecedu nebo chce tiskárnu používat jako standardní tiskárnu pro počítač IBM PC, nemusí tyto požadavky zadávat po každém zapnutí příslušnými řídicími posloupnostmi. Uvnitř v tiskárně jsou umístěny čtyři přepínače, kterými uživatel může předvolit některé důležité požadavky. Tiskárna se po zapnutí sama nastaví do režimu podle přepínačů předvolby.

Tiskárna také umí zjistit, zda je nebo není stisknuto některé ze dvou tlačítek na vrchním panelu tiskárny. Tam jsou také umístěna dvě světýlka, která můžeme rozsvěcet nebo zhasínat.

Shrnutí:

Naše jehličková tiskárna nám umožňuje rozjet a zastavit barvicí pásku, posunout tiskací hlavu o krok doleva nebo doprava, posunout papír o krok nahoru nebo dolů a vytisknout naráz libovolnou kombinaci devíti teček nad sebou pomocí devíti jehel v tiskací hlavě. Pípák s obvodem 8253 nám umožňuje vydávat tóny zvolené výšky a délky. Můžeme číst stav 4 přepínačů předvolby umístěných uvnitř tiskárny a stav dvou tlačítek na vrchním panelu a rozsvěcet dvě světýlka.

3.3. Podrobné zadání úlohy

Tato kapitola má ukázat zákazníkům a programátorům, jak zadávat úlohu. Jsou zde popsány všechny funkce jehličkové tiskárny slučitelné s EPSON FX-85.

V kapitole 3.1. jsme se zabývali tím, co by měla umět každá jehličková tiskárna. Nedomluvili jsme se však na tom, co má který řídicí znak nebo posloupnost přesně znamenat.

Každý jen trochu ješitný zákazník by se rád pyšnil lepší tiskárnou, než mají všichni ostatní. Každý jen trochu ješitný konstruktér nebo programátor by rád předvedl pronikavost svého rozumu a oslnivost svého talentu a zároveň se splněním zakázky by si chtěl vysloužit nesmrtelnost.

Ale běda! Uživatel zpravidla nebývá obdivovatelem lidského důmyslu a zbožňovatelem technických novinek. Uživatel touží po zařízení důvěrně známém z dřívějška, po přístroji zcela pro-

stoduchém a tím i nezáludném, po tiskárně, která ho nebude obtěžovat nezvyklými požadavky, která mu bude sloužit a nikoli vládnout, kterou bude moci používat a nikoli učit se ji používat. K dobré pracovní pohodě uživatele patří, cítí-li se mocným pánem svých strojů, má-li pocit, že svým strojům dokonale rozumí a mentálně vysoce převyšuje všechny, kteří je vymýšleli. Tato hlediska se ovšem odrážejí i v nákladech na zaškolení obsluhy a v možnosti začlenit nový přístroj do starého systému.

Proto moudrý zákazník, konstruktér nebo programátor poskytuje uživatelům to, co chtějí, a svůj věhlas šíří jen v úzkém okruhu zasvěcenců, které zdaleka nezajímá, **co** nová tiskárna všechno umí, ale **jak** je to uděláno.

Pokusme se zjistit, na jakou tiskárnu jsou uživatelé zvyklí. Lze se domnívat, že nejlépe vyhoví některá z tiskáren EPSON japonské firmy SEIKO EPSON Corporation. Jako vzor pro volbu řídicích znaků a řídicích posloupností naší tiskárny zvolíme jeden z novějších typů, který je slučitelný jak se staršími tiskárnami EPSON, tak s tiskárnami IBM PC matrix printer (to jsou jehličkové tiskárny, které používá IBM PC, jeden z nejrozšířenějších osobních počítačů na světě). Navíc je tento typ slučitelný i se všemi epigony tiskáren EPSON. Jedná se o EPSON FX-85 z roku 1985. Novější tiskárny EPSON totiž nabízejí příliš velké množství různých sad znaků, než abychom vystačili se skromným rozsahem paměti, který máme k dispozici. Z novějších tiskáren však můžeme převzít aspoň některé užitečné řídicí posloupnosti. Kromě tohoto rozšíření připustíme i některé kombinace druhů písma, které FX-85 nepřipouští (zjednoduší se tím náš program).

Nyní můžeme upřesnit, co všechno má tiskárna umět a jak to má dělat.

Tiskárna se neřídí jen přijímanými znaky. Činnost tiskárny může ovlivnit i obsluha, a to pomocí tlačítek na vrchním panelu. A tiskárna zase sděluje obsluze vše potřebné pomocí pípáku a světýlek. Budeme se tedy zabývat tím, jak se dorozumívá člověk se strojem. Dostáváme se k velmi závažnému tématu, které snad v blízké budoucnosti upoutá nejen zájem programátorů, ale i sociologů a psychologů.

Každý programátor by si měl uvědomit, že jeho programy mají nějakým způsobem (přímo nebo zprostředkovaně) sloužit lidem. A měl by považovat za svou lidskou povinnost, aby nedopustil opak.

Zatím se však až příliš často setkáváme s programy podobnými tomu, který už při zahájení své činnosti vyžaduje příkaz 3EH97EV2ET. A kdo si předem podrobně neprostuduje několik set stránek příručky v angličtině, nemá šanci domluvit se s tímto programem. Jsou programy (a autora v tuto chvíli hryže svědomí), které nemilosrdně trestají kdejakou drobnou chybu obsluhy znehodnocením výsledků několikahodinové práce. Jiné programy jsou zase záludné a číhají na chvilkovou nepozornost uživatele, aby mu vyvedly nějakou zlomyslnost. Mezi uživateli programu TECO se

rozšířila hra, při které každý hráč zadá jako příkaz své jméno. Vyhrává ten, kdo způsobí největší škodu.

Programátor by si měl uvědomit, že je zodpovědný za chování programu, že podle chování programu bude on sám hodnocen. Přívětivé, přátelské chování k uživateli je zásada každého moderního profesionálního programátora. Tuto zásadu ctil i Frank Borland, když psal svůj slavný TURBO PASCAL, a nejen že vydělal peníze, ale získal si obdiv a úctu na celém světě.

Uveďme několik zásad, kterým má dobrý program vyhovovat:

Jednoduchost. Na jednoduchý pokyn uživatele se vykoná jednoduchá činnost.

Prostoduchost. Profesionální programátor nikdy nedává uživateli najevo svou profesionální převahu (nemá to ani zapotřebí). K používání programu musí stačit co nejméně znalostí. Program a uživatel by se měli domlouvat v jazyce uživatele.

Poslušnost. Program se přesně řídí příkazy uživatele. Zásadně nedělá nic víc a nic míň, než od něj uživatel žádá a očekává.

Rozumnost. Program vychází vstříc potřebám uživatele a nabízí mu všechny užitečné služby. Nenabízí takové služby, které nikdo nepotřebuje. Program se řídí logikou uživatele, nikoli logikou počítače.

Bezpečnost. Program chrání uživatele proti následkům chyb uživatele. Buď existuje příkaz, kterým se uživatel může vrátit do stavu před vykonáním chybného příkazu, nebo si program u nebezpečných příkazů ověřuje, zda je uživatel chce doopravdy provést.

Ochota. Program by měl uživateli nabízet své služby. Používat program a listovat přitom v příručce je projev barbarství, zaostalosti a zpátečnictví programátora.

Vraťme se k naší tiskárně a pokusme se popsat, jaké služby by měla poskytovat obsluze. Především by neměla klást na obsluhu velké požadavky. V podstatě by mělo stačit zapnutí vypínačem k tomu, aby tiskárna mohla začít tisknout. Ale je dobře dát to nějak najevo. K tomu poslouží jedno ze světýlek (označené nápisem "HOTOVO"), které bude oznamovat, že tiskárna může přijmout znak z počítače. Navíc pípák zahraje při zapnutí nějakou melodii, která sdělí obsluze, že tiskárna je schopna pracovat. Takový způsob činnosti, kdy tiskárna přijímá znaky z počítače a tiskne je, nazýváme spřažený režim. Tiskárna je v takovém případě spřažena s nadřízeným počítačem.

Někdy je nutné přerušit tisk (např. aby obsluha mohla srovnat papír nebo barvicí pásku). K tomu poslouží tlačítko "RUČNÍ". Při stisknutí tohoto tlačítka tiskárna přejde do ručního režimu činnosti a rozsvítí světýlko "RUČNÍ". V ručním režimu bude možné posouvat papír stisknutím tlačítka "POSUN". Když bude tlačítko "POSUN" stisknuto krátce, posune se papír o řádek, když obsluha tlačítko chvíli podrží, posune se papír na začátek další stránky.

Když tiskárna zjistí, že už nemá papír, přepne se sama do ručního režimu a zahraje písničku s významem "došel papír". Obsluha pak může založit nový papír a pokračovat v tisku.

A jak se tiskárna vrátí do spřaženého režimu? Nejjednodušší bude, když obsluha znovu stiskne tlačítko "RUČNÍ".

Naše tiskárna má sloužit především českým a slovenským uživatelům, a proto bychom měli její sadu znaků rozšířit o sadu českých a slovenských písmen s diakritickými znaménky. Pokud se konstruktéři našich počítačů drželi nějaké normy pro českou abecedu, používali většinou kód KOI-8čs. Novější počítače třídy IBM PC sice tento kód nepoužívají, ale mají programy, které píší české texty stejným způsobem, jakým se kreslí obrázky. Takže vůbec nevyžadují, aby tiskárna měla sadu českých znaků. Proto použijeme sadu znaků kódu KOI-8čs, kterou si zvídavý čtenář může prohlédnout v tabulce 3.1b.

Jehličkové tiskárny obvykle nabízejí několik typů písma. Buď píší **ledabyle** a rychle, nebo **pečlivě** a pomalu. Dále mohou tisknout různě široká písmena, takže na řádku dlouhou 8 krát 25,4 mm se vejde 40 až 160 znaků. Pokud snad čtenáře zaráží podivná míra 25,4 mm, dovoluje si autor upozornit, že se jedná jen o zákonné vyjádření nezákonné měřicí jednotky "palec". Používání slova "palec" je sice pohodlné, nicméně ilegální.

V kapitole 3.2 jsme si ukázali, jak se znaky skládají z černých teček. Černé tečky zakreslujeme do okénka rozděleného na 24 krát 12 políček.

Chceme-li psát pečlivě, můžeme použít všechna políčka v okénku. Tečka je o něco větší než políčko, takže její okraj zasahuje do sousedních políček. Jestliže umístíme tečky do sousedních políček, slijí se do souvislé černé skvrny. Takto vytvořená písmena jsou skoro tak pěkná, jako písmena psacího stroje. Při tisku takových písmen se však tiskací hlava musí pohybovat pomalu, aby stačila vytisknout všech 12 sloupců vedle sebe. A již jsme si řekli, že všech 24 linek je třeba tisknout pomocí několika průchodů hlavy. Naše tiskárna má 9 jehel, proto bude tisknout 24 linek na 4 průchody. Obvyklé znaky, jako jsou písmena a číslice, se vejdou do 18 linek a vytisknou se tedy jen dvěma průchody.

Chceme-li zrychlit pohyb hlavy a navíc omezit počet průchodů na jeden až dva, musíme okénko vyplňovat tečkami ob linku a ob sloupec, takže vyplníme jenom 6 krát 12 políček. Omezíme-li se jen na vrchních 9 linek, vytiskne se znak jediným průchodem hlavy. Použijeme-li i tři nejspodnější linky v okénku, bude se tisknout nadvakrát.

Jak ale tiskárna pozná, která políčka má vyčernit, když se tiskne třeba písmeno A, a která při B? K tomu má tiskárna ve své paměti tabulku, nazývanou **generátor znaků**. Tabulka má tolik položek, kolik různých kódů mohou mít tištěné znaky. Položky jsou v generátoru uspořádány podle kódů, aby se pro každý znak dala

snadno a rychle najít správná položka. Každá položka v generátoru obsahuje několik množin, jejichž prvky označují ta políčka, která je třeba vyčernit.

Šířka znaku je určena počtem použitých sloupců. Základním druhem písma je **pica** [čti pajka]. Používáme pro něj 12 sloupců. Těchto znaků se vejde 80 na řádek. Typ pica přibližně odpovídá našemu typu cicero, který je u nás užíván jako základní typografická míra.

O něco užší je písmo **élite**, 10 sloupců. Na jeden řádek se vejde 96 znaků tohoto typu.

Chceme-li tisknout dlouhé řádky na úzký papír, můžeme písmo zhustit: picu na 7 sloupců (137 znaků na řádek), élite na 6 sloupců (160 znaků na řádek).

Někdy je třeba psát výrazné nápisy. Toho se dosahuje **rozšířením** písma na dvojnásobek - každý sloupec se tiskne dvakrát. Rozšířená pica má například 40 znaků na řádek.

Další požadavek na šířku písmen vyplývá z toho, že každé písmeno je jinak široké. Například m je širší než i. Obvykle to lze zanedbat, projevuje se to různě velkými mezerami mezi písmeny a to málokdy vadí. Tiskárny však také umožňují tisknout každý znak jen na tolik sloupců, kolik skutečně zabírá, a mezi znaky se pak vkládá vždycky stejná mezera. Takovému způsobu tisku se říká **proporcionální**.

Dalším užitečným druhem písma je **tučný tisk (toto je tučný tisk)**. Dosáhneme ho tak, že do okénka zakreslíme vždy dva stejné sloupce vedle sebe (a zpomalíme tím pohyb hlavy) nebo dvě stejné linky pod sebe (a budeme tisknout nadvakrát). Kombinací obou způsobů dosáhneme pečlivého tučného tisku.

I obrázky lze tisknout pečlivěji nebo ledabyleji, hustěji nebo řidčeji. Bystrý čtenář si již mohl spočítat, že znaky se tisknou buď v hustotě 120 sloupců na 25,4 mm (pečlivě) nebo 60 sloupců na 25,4 mm (ledabyle). U obrázků máme větší požadavky: kromě hustoty 60 a 120 sloupců na 25,4 mm lze na lepších tiskárnách vytisknout i 240 sloupců na 25,4 mm (a horší tiskárny, které to nedokážou, musí aspoň přijímat příslušnou řídicí posloupnost). Pro opisování obrazovky displeje se užívá hustot 80 nebo 90 sloupců na 25,4 mm a pro dosažení stejné svislé a vodorovné hustoty při tisku grafů se užívá hustot 72 sloupců na 25,4 mm nebo dvojnásobné hustoty 144 sloupců na 25,4 mm.

Při tisku obrázků je výhodné, aby měl řádek jen 8 nebo 9 linek. Řídicí posloupnost je složena z množin, které popisují černá a bílá políčka v řádku. Každá množina popisuje osm nebo devět políček pod sebou. To právě odpovídá sloupci jehel v hlavě. V případě, že množina popisuje osm políček, můžeme ji zobrazit v jednom bajtu (devítiprvková množina se proto zadává nadvakrát). Takový řádek složený z osmi nebo devíti linek se dá vytisknout na jeden průchod hlavy, takže se tím šetří čas.

Tiskárna také musí umět posouvat papír. Jeden krok motoru posune papír zpravidla o 1/3 nebo o 1/2 linky, aby se posunem

papíru o 1 krok dostaly jehly v hlavě do polohy mezi linky, které byly vytištěny před posunutím papíru. Obvyklá hustota linek u jehličkových tiskáren je 72 linek na 25,4 mm. Motor tedy musí udělat 216 nebo 144 kroků na 25,4 mm.

Již jsme se zmínili, že každý znak může být sestaven až z dvanácti linek. Tomu odpovídá i hustota řádků. Obvykle se tedy píše 6 řádků na 25,4 mm. Pokud bychom chtěli tisknout co nejhustěji, je třeba počítat s devíti linkami na řádek. Všechna písmena, číslice a znaky používané v textu se totiž vejdou na 9 linek a tisknou se na jeden průchod ledabyle nebo na dva průchody pečlivě. Posunu papíru o devět linek odpovídá hustota řádkování 8 řádků na 25,4 mm.

Řádkování je možné nastavit i na nějakou jinou hustotu vyjádřenou buď v linkách (tj. 25,4 / 72 mm), nebo v krocích motoru (tj. 25,4 / 216 mm). Jestliže tiskárna používá pro posun papíru krok 25,4 / 144 mm, musí nastavovat papír do přibližných poloh vypočtených podle řádkování zadaného v 25,4 / 216 mm.

Do tiskáren se používá dlouhý pás papíru, poskládaný po stránkách jako dětské leporelo a perforovaný na hranicích stránek jako toaletní papír, aby se dal snadno roztrhat na jednotlivé stránky.

Pokud se při řádkování dosáhne konce stránky, hrozí nebezpečí, že by se tisklo do místa, kde je papír přeložený a perforovaný. Proto tiskárna musí včas posunout papír na začátek následující stránky. Existuje řídicí posloupnost, která nastavuje výšku stránky, a jiná řídicí posloupnost, která nastavuje, kolik papíru se má přeskočit mezi koncem jedné stránky a začátkem další.

Je však také možné odstránkovat dřív, než se popíše celá stránka. K tomu slouží řídicí znak stránkování - tiskárna posune papír na začátek další stránky. Předpokládá se, že při zapnutí tiskárny bude papír nastaven na začátek stránky.

Často je také potřeba nastavit hlavu do předem zvolené polohy v řádce (aby se dalo tisknout do sloupců pod sebe) nebo na předem zvolenou řádku ve stránce. K tomu slouží řídicí znaky pro vodorovnou a svislou tabulaci a řídicí posloupnosti pro nastavování vodorovných a svislých tabulačních zarážek.

Dostane-li tiskárna příkaz k vodorovné tabulaci, najde v seznamu vodorovných tabulačních zarážek nejbližší zarážku směrem vpravo od místa, kam se vytiskne předchozí znak. Následující znak se bude tisknout do polohy určené zarážkou. Při zapnutí tiskárny jsou vodorovné tabulační zarážky nastaveny na pozici devátého znaku v řádce, sedmnáctého znaku v řádce, 25., 33., 41. atd. znaku v řádce, vždy po osmi znakových pozicích. Toto počáteční nastavení lze změnit řídicí posloupností, která obsahuje nové znakové pozice pro tabulační zarážky.

Dostane-li tiskárna příkaz ke svislé tabulaci, najde v seznamu svislých tabulačních zarážek nejbližší zarážku směrem dolů od místa, kam se vytiskne předchozí řádka. Následující znak

se bude tisknout do řádky určené zarážkou. Při zapnutí tiskárny jsou svislé tabulační zarážky nastaveny na všechny řádky ve stránce, takže řídicí znak pro svislou tabulaci způsobí totéž jako řídicí znak pro řádkování. Toto počáteční nastavení lze změnit řídicí posloupností, která obsahuje nové řádkové pozice pro tabulační zarážky.

Tiskárna také podporuje psaní textů. Je možné nastavit levý a pravý okraj řádku, takže buď můžeme využít celou šířku papíru, anebo třeba jen šedesát znaků na řádek, když se jedná o nějakou úřední listinu. Můžeme si třeba udělat široký levý okraj, aby se vytištěné stránky daly snadno svázat.

Tiskárna musí umět zarovnat řádky podle levého nebo pravého okraje. Tiskárna také musí umět umístit řádku doprostřed mezi okraje.

Složitější případ nastane, když se bude tisknout hodně dlouhá řádka. Písmena se budou skládat do řádky tak dlouho, až se řádka naplní. Jak budeme tisknout dál? František Palacký by asi hledal řešení v duchu výroku: "tiskli jsme před koncem řádky, budeme tisknout i po něm". Kam? Odpověď je zřejmá: na začátek další řádky. Jenže v tom je háček - často se nám přeplní řádka zrovna uprostřed slova. A protože neexistuje jednoduchý návod na dělení slov, nemůžeme slovo na konci řádky rozdělit. Proto tiskárna musí takové slovo, které přesahuje okraj řádky, "sbalit".

Tiskárna bude balit slova podle tohoto návodu: Jestliže se naplní řádka a právě přijatý znak by se již na řádku nevešel, najde se poslední mezera nebo řídicí znak pro vodorovnou tabulaci (HT). Řádka se vytiskne jen k této mezeře nebo ke znaku HT, odřádkuje se a znaky za mezerou nebo za znakem HT se umístí na následující řádku. Tento postup vychází z předpokladu, že každá dvě slova v textu jsou oddělena aspoň jednou mezerou nebo znakem HT. Když se tiskne taková přeplněná řádka (zkrácená o poslední nedokončené slovo), lze ji zarovnat podle obou okrajů. Při tom se zvětšují mezery uvnitř řádky tak, aby byla řádka správně dlouhá od kraje do kraje.

Teď, když jsme se seznámili s tím, co všechno má tiskárna umět, uvedeme seznam všech řídicích znaků a posloupností a jejich význam.

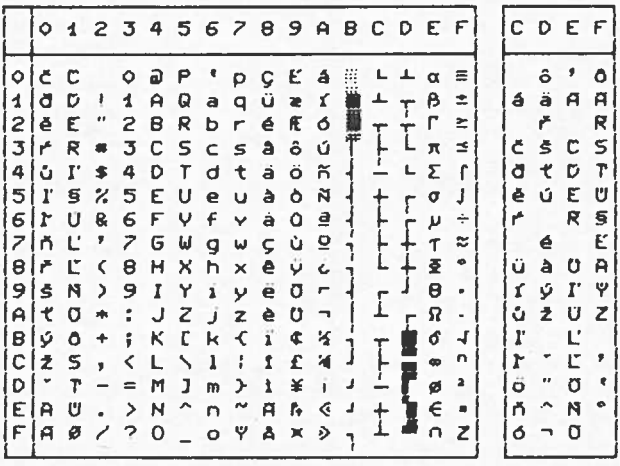
Některé řídicí posloupnosti mají dvojí význam. To závisí na tom, zda tiskárna zrovna pracuje podle zvyklostí IBM nebo EPSON. Po zapnutí se totiž tiskárna podívá na jeden z přepínačů předvolby umístěných uvnitř tiskárny. Tímto přepínačem se dá trvale nastavit, podle jakých zvyklostí se bude tiskárna chovat.

Řídicí znaky se označují tradičními anglickými zkratkami, jako třeba ESC, HT, LF apod. Podobně jako tištitelné znaky mají i řídicí znaky své umístění v kódové tabulce a podle toho jim jsou přiřazeny číselné kódy (ty budeme používat v kapitole 4).

Uveďme si proto přehled řídicích znaků a jejich kódů:

NUL	00h		LF	0Ah		SO	0Eh		DC3	13h		SP	20h
BEL	07h		VT	0Bh		SI	0Fh		DC4	14h		DEL	7Fh
BS	08h		FF	0Ch		DC1	11h		CAN	18h			
HT	09h		CR	0Dh		DC2	12h		ESC	1Bh			

Kromě řídicích znaků se v řídicích posloupnostech mohou vyskytnout písmena, číslice a jiné tištitelné znaky, čísla a množiny. Tištitelné znaky budeme uvádět v apostrofech (např. 'A' nebo 'x1'), čísla budeme označovat symbolicky n, n1, n2 atd. a množiny budeme označovat symbolicky m, m1, m2 atd. Ve vysvětlujícím textu bude uvedeno, jakých hodnot mohou čísla nebo množiny nabývat. V našem seznamu budeme u každého řídicího znaku nebo posloupnosti uvádět anglickou zkratku, český název a vysvětlení významu. Vysvětlení se bude opírat o předchozí výklad.



a) sada znaků IBM b) část KOI-8čs

Tab. 3.1 (v tabulce b) jsou odlišnosti KOI-8čs proti IBM)

Seznam řídicích znaků a posloupností:

3.3.1. Styk s obsluhou

BEL

Zahraj na pípák. Tento znak slouží např. k přivolání obsluhy.

ESC '8'

Přestaň se zajímat o to, zda máš dost papíru.

Tento znak je užitečné použít, chceme-li tisknout na jednotlivé listy papíru. Jinak totiž tiskárna přestane tisknout a bude hlásit, že došel papír ještě před dokončením tisku na každé stránce.

ESC '9'

Začni se opět zajímat o to, zda máš dost papíru.

3.3.2. Styk s počítačem

ESC '>'

U údajů, které budou následovat, nastavuj nejvýznamnější bit na hodnotu 1.

Některé počítače totiž vysílají údaje do tiskárny jen po 7 vodičích. Tyto vodiče přenášejí 7 nejméně významných bitů. Proto je třeba předem nastavit hodnotu osmého, nejvýznamnějšího bitu. Tiskárna vždy přidá tuto hodnotu k hodnotám sedmi přijatých bitů a takto získanou osmibitovou hodnotu bude tiskárna považovat podle potřeby buď za znak, nebo za číslo, anebo za množinu.

ESC '='

(podle zvyklostí EPSON)

U údajů, které budou následovat, nastavuj nejvýznamnější bit na hodnotu 0.

ESC '#'

Přestaň měnit nejvýznamnější bit u přijímaných údajů a přijímej jej opět z počítače.

3.3.3. Vnitřní funkce tiskárny

ESC '@'

Přejdi do stavu jako po zapnutí síťovým vypínačem.

ESC 'U' n
Když n = 1, tak tiskni jen zleva doprava.
Když n = 0, tak tiskni oběma směry.
Příklad obousměrného tisku:

emememememememememememememememememememememememememe

Příklad jednosměrného tisku:

emememememememememememememememememememememememememe

Tiskací hlava obvykle tiskne bez ohledu na směr svého pohybu - např. první řádek zleva doprava, druhý zprava doleva, pak zase zleva doprava atd., aby byla hlava co nejvíce využita a aby se co nejméně pohybovala naprázdno. Vlivem různých nepřesností však dochází k tomu, že sloupce teček v různých řádkách, které mají být pod sebou, se poněkud posunou. Takové rozházené tečky pak kazí vzhled tučného tisku nebo tisku obrázků. Jestliže tiskárna bude tisknout jen zleva doprava, bude se sice hlava pohybovat zprava doleva naprázdno, ale nepřesnosti se neprojeví a tím se podstatně zlepší vzhled tisku.

ESC '<'
Najeď hlavou na levý okraj a vytiskni následující řádek zleva doprava.

ESC 's' n
Když n = 1, tak tiskni pomaleji a pěkněji, např.:

//

Když n = 0, tak tiskni opět co nejrychleji, např.:

//

ESC 'i' n
Když n = 1, tak každý přijatý znak okamžitě vytiskni a ukaž, cos napsala (tím způsobem, že vysuneš papír o kousek nahoru).
Když n = 0, tak tiskni zase každou celou řádku najednou.

DEL
Smaž poslední znak v řádce.
Tiskárna přijímá znaky, ale obvykle je netiskne okamžitě, nýbrž teprve až se dokončí celá řádka. Není-li poslední přijatý znak dosud vytištěn, odstraní se z paměti tiskárny, takže se už nevytiskne.

CAN

Smaž celou poslední řádku. Tiskárna vymaže ze své paměti všechny znaky, které přijala a dosud nevytiskla.

NUL

Nic nedělej. Tento znak je dědictvím dávných dob, kdy se hojně používaly děrné pásky. Začátek a konec pásky neobsahuje žádná data (slouží k tomu, aby bylo možné založit pásku do snímače nebo uchytit ji v navíječi). Když se vypisoval obsah pásky na tiskárně, přečetl se na všech prázdných místech na pásce znak NUL, který oznamoval tiskárně: tady je prázdné místo na pásce, nic netiskni.

3.3.4. Řádkování

LF

Vytiskni řádku a posuň papír o jednu řádku. Následující řádka pak bude začínat vpravo od konce právě vytištěné řádky, ale o řádku níž, např.:

omomomomomomomomom

omomomomomomomomom

Proto se před znakem LF obvykle používá znak CR, aby následující řádka začínala na levém okraji.

ESC LF

Vytiskni řádku a posuň papír o jednu řádku zpět.

ESC '0' ('0' je číslice nula)

Nastav rozteč řádek na 1/8 krát 25,4 mm. Obvyklá rozteč řádek je 1/6 krát 25,4 mm. Po vykonání posloupnosti ESC '0' budou následující řídicí znaky LF a ESC LF řádkovat jen o 1/8 krát 25,4 mm.

ESC '1'

Nastav rozteč řádek na 7/72 krát 25,4 mm.

ESC '2'
(podlě zvyklostí EPSON)

Nastav rozteč řádek na 1/6 krát 25,4 mm.

ESC 'A' n
(podle zvyklostí EPSON)

Nastav rozteč řádek na n/72 krát 25,4 mm.

ESC '2'
(podle zvyklostí IBM)

Nastav rozteč řádek na hodnotu připravenou pomocí posloupnosti ESC 'A' podle zvyklostí IBM. Jestliže dosud nebyla přijata

posloupnost ESC 'A', nastaví se rozteč řádek na 1/6 krát 25,4 mm (tj. stejně jako podle zvyklostí EPSON).

Tato řídicí posloupnost je diluviální obludou mezi řídicími posloupnostmi, s níž se setkáváme ještě v roce 1985 zcela v rozporu se zákony přirozeného výběru. Byla zřejmě zavedena firmou IBM za účelem vyřazení mentálně slabších uživatelů z řad svých zákazníků nebo snad jako duševní cvičení pro neobyčejně inteligentní jedince. Vzhledem k tomu, že tato posloupnost dělá složitě to, co lze udělat i jednoduše, uvádíme ji pouze z důvodů slučitelnosti (a z týchž důvodů ji zřejmě používala i firma EPSON ještě u tiskárny FX-85; u modelu FX-800 však již i firma EPSON raději použila zdravého rozumu).

Autor, maje na zřeteli svůj zájem o přízeň publika, snažně prosí čtenáře, kteří napoprvé neporozuměli této zkamenělině, aby se o to ani nesnažili.

ESC 'A' n
(podle zvyklostí IBM)

Připrav rozteč řádek n/72 krát 25,4 mm pro použití v ESC '2' podle zvyklostí IBM. Viz též ESC '2'.

ESC '3' n

Nastav rozteč řádek na n/216 krát 25,4 mm.

ESC 'J' n

Vytiskni řádek a posuň papír vpřed o n/216 krát 25,4 mm. Tato posloupnost nemá vliv na rozteč řádkování, které provádějí znaky LF a ESC LF (což je zřejmé z předchozí věty).

ESC 'j' n

Vytiskni řádek a posuň papír zpět o n/216 krát 25,4 mm.

3.3.5. Stránkování

FF

Posuň papír na začátek další stránky.

ESC FF

Posuň papír zpět na začátek stránky.

ESC '4'
(podle zvyklostí IBM)

Tato posloupnost sděluje, že papír v tiskárně byl právě ručně nastaven na začátek stránky.

ESC 'C' n

Když n se nerovná 0, tak tato posloupnost sděluje tiskárně, že stránka papíru je dlouhá n řádek (včetně řádek, které leží na hranici stránek a které tiskárna normálně přeskakuje).

ESC 'C' 0 n (0 je číslo nula)

Tato posloupnost sděluje tiskárně, že stránka papíru je dlouhá n krát 25,4 mm. (Obvykle se používá papír se stránkami dlouhými 11 nebo 12 krát 25,4 mm).

ESC 'O' ('O' je písmeno velké O)

Zruš přeskok perforace na hranicích stránek. Po zapnutí tiskárny síťovým vypínačem je nastaven přeskok na 25,4 mm nebo na nulu podle přepínače předvolby. Podle toho se buď vynechává 6 řádek na spodním okraji každé stránky, nebo se tiskne souvisle i přes perforaci. Dotyčná posloupnost nastavuje délku přeskoku na nulu.

ESC 'N' n

Vynechávej n řádek na spodním okraji každé stránky, aby se netisklo přes perforaci na hranici stránek.

3.3.6. Tisk obrázků

ESC 'K' n1 n2 m1 m2...

Tiskni ledabyle obrázek.

Množiny m1, m2, ... jsou množiny černých teček ve sloupcích na řádku. Každý sloupec může obsahovat až 8 černých teček. V množině může být až 8 různých prvků a každému z těchto osmi prvků přísluší jedno z osmi míst ve sloupci.

Pilný čtenář si snad ještě pamatuje z kapitoly 2.5, že kvůli zobrazení v počítači musí být prvky množiny uspořádány. Prvky množiny m jsou uspořádány podle umístění teček ve sloupci zdola nahoru.

Nyní můžeme množinu teček popsat typem:

```
Typ
  tečky = (t1, t2, t3, t4, t5, t6, t7, t8);
  sloupec_teček = množina nad tečky;
```

Z kapitoly 3.2 si pozorný čtenář jistě pamatuje, že tiskací hlava naší tiskárny má 9 jehel. Dost možná mu vytanula na mysli otázka, kterých 8 jehel bude použito pro tisk obrázku a která jehla zůstane nevyužita? Věz, zvídavý čtenáři, že bude použito vrchních osmi jehel a že nejspodnější jehla bude zahálet.

Nyní se vraťme k obsahu řídicí posloupnosti. Bystrého čtenáře asi sžírá pochybnost, zda tiskárna pozná, kolik množin posloupnost obsahuje, a tedy kolik sloupců má vytisknout. Tento počet udávají čísla n1 a n2 podle vzorce

Počet_Množin = n1 + 256 * n2,

kde znaménko * označuje násobení.

A nakonec ještě zbývá upřesnit, co znamená, že se bude tisknout ledabyle: to znamená, že se bude tisknout 60 sloupců na 25,4 mm.

Např.: posloupnost

ESC 'K' 10 0
[t7, t1]
[t7, t2]
[t7, t2, t1]
[t7, t3]
[t7, t3, t1]
[t7, t3, t2]
[t7, t3, t2, t1]
[t7, t4]
[t7, t4, t1]
[t7, t4, t2]

vytiskne:

ESC 'L' n1 n2 m1 m2...

Tiskni pečlivě obrázek. Posloupnost má stejný význam jako ESC 'K', ale sloupce se tisknou hustěji - 120 na 25,4 mm.

Např.: posloupnost ESC 'L' 10 0 [t7, t1]...[t7, t4, t2]

vytiskne:

ESC 'Y' n1 n2 m1 m2...

Tiskni rychle a ledabyle pečlivě připravený obrázek. Posloupnost má stejný význam jako ESC 'L'.

Používá se při kontrolních tiscích obrázků - tisk se zrychlí za cenu zhoršení vzhledu. Podobného výsledku lze dosáhnout i použitím posloupnosti ESC 'K', ale běda: tím by se obrázek rozšířil na dvojnásobek! Proto v případě posloupnosti ESC 'Y' tiskárna vynechává každý druhý sloupec a tím zůstane obrázek správně široký. Až si ho uživatel prohlédne a shledá ho hodn m pečlivého tisku, jednoduše nahradí znak 'Y' v řídicí posloupnosti znakem 'L' a tiskárna mu pak vytiskne vzhledný obrázek.

Např.: posloupnost ESC 'Y' 10 0 [t7, t1]...[t7, t4, t2]

vytiskne:

ESC 'Z' n1 n2 m1 m2...

Tiskni pečlivě obrázek a předstírej, že umíš tisknout ještě pečlivěji než ve skutečnosti. Tato posloupnost se opět podobá posloupnosti ESC 'K', tiskne se však s hustotou 120 sloupců na 25,4 mm a ještě k tomu vždy dva sloupce přes sebe.

Tím se vytvoří klamný dojem, že se tiskne 240 sloupců na 25,4 mm. Takové hustoty tisku a přiměřené přesnosti totiž dosahují nejlepší jehličkové tiskárny včetně FX-85. My ovšem v tak kvalitní mechanickou část tiskárny nedoufáme, proto se z toho pokusíme vyzout uvedeným způsobem.

Např.: posloupnost ESC 'Z' 10 0 [t7, t1]...[t7, t4, t2] vytiskne:

ESC '*' n0 n1 n2 m1 m2...

Tiskni obrázek ve zvolené hustotě. Tato posloupnost se opět nápadně podobá posloupnosti ESC 'K', liší se však prvkem n0. Ostatní prvky posloupnosti mají zase stejný význam jako u předchozích posloupností. Prvek n0 určuje hustotu tisku. Hodnoty n0 = 4 až 7 jsou zavedeny pro ty případy, kdy se má zachovat správný poměr vodorovné a svislé velikosti obrázku (např. když se opisuje obrazovka displeje):

n0 = 0...60 sloupců na 25,4 mm (normální tisk)
n0 = 1...120 sloupců na 25,4 mm (pečlivý tisk)
n0 = 2...120 sloupců na 25,4 mm (ledabylý tisk, viz ESC 'Y')
n0 = 3...240 sloupců na 25,4 mm (tisk s klamnou hustotou)
n0 = 4...80 sloupců na 25,4 mm (pro opis obrazovky displeje)
n0 = 5...72 sloupců na 25,4 mm (stejná hustota jako svisle)
n0 = 6...90 sloupců na 25,4 mm (pro opis obrazovky displeje)
n0 = 7...144 sloupců na 25,4 mm (obdoba n0 = 5)

ESC '^' n0 n1 n2 m1 m2 m3 m4...
(podle zvyklostí EPSON)

Tiskni obrázek na 9 linkách. Tato posloupnost se podobá posloupnosti ESC '*', obsahuje však dvojnásobný počet množin, tj.

2 * (n1 + 256 * n2).

Každý sloupec, který se bude tisknout, je totiž určen dvojicí množin, takže se vlastně jedná o posloupnost dvojic množin. První množina z dvojice popisuje stejně jako dřív 8 teček, druhá množina popisuje devátou, nejspodnější tečku ve sloupci (Pozornému čtenáře snad bude zajímat, že tato informace je uložena v nejvýznamnějším bitu druhé množiny).

ESC '?' c n

Změň hustotu tisku pro posloupnosti ESC 'K', ESC 'L', ESC 'Y', ESC 'Z'. Hodnota c určuje, která řídicí posloupnost bude nadále ovlivněna (c je písmeno 'K', 'L', 'Y' nebo 'Z'). Hodnota n určuje hustotu, s jakou se bude tisknout. Nabývá týchž hodnot a má stejný význam jako n0 v posloupnosti ESC '*'.

3.3.7. Generátory znaků

ESC '^' c
(podle zvyklostí IBM)

Tiskni znak c. Tato posloupnost je vhodná k tisku libovolných znaků.

Bystrý čtenář již možná tuší, že generátor znaků v sobě skrývá víc, než jsme schopni z něj vyloudit obvyklými postupy. Vtírá se otázka, co by se stalo, kdyby si tiskárna po přijetí řídicího znaku nevšimla, že je to řídicí znak, a místo obvyklé činnosti se pokusila ho vytisknout. Vždyť generátor znaků musí v sobě něco ukrývat i pod kódy řídicích znaků! A co ukrývá? To zví dychtivý čtenář, když popatří do tabulky 3.1a. Např. posloupnost ESC '^' BEL vytiskne znak ň.

ESC '\' n1 n2 c1 c2 c3...
(podle zvyklostí IBM)

Tiskni znaky c1, c2, c3 atd. Počet znaků, které má tato řídicí posloupnost vytisknout je

n1 + 256 * n2,

kde * je znaménko pro násobení.

Tato posloupnost - podobně jako ESC '^' - je vhodná k tisku těch znaků, které jsou v generátoru skryty pod znaky řídicími.

ESC '6'

Přestaň považovat řídicí znaky v druhé polovině kódové tabulky za řídicí a normálně je tiskni pomocí generátoru. Jedná se o znaky s kódy 080h až 09Fh. První polovina tabulky obsahuje i nadále řídicí znaky.

Čtenáři, který považuješ kódy 080h a 09Fh za nerozluštitelné šifry cizí zpravodajské služby, věz, že náležité vysvětlení najdeš v kapitole 2.6.

ESC '7'

Považuj znaky s kódy 080h až 09Fh opět za řídicí. Tato posloupnost ruší účinek posloupnosti ESC '6'.

ESC 'I' n
(podle zvyklostí EPSON)

Když n = 1, tak přestaň považovat řídicí znaky NUL a CAN a řídicí znaky v druhé polovině kódové tabulky za řídicí a normálně je tiskni pomocí generátoru. Znaky, které jsou v kódové tabulce roztroušeny mezi řídicími znaky, tiskni také. Jedná se o znaky s kódy 000h až 006h, 010h, 011h, 013h, 015h až 01Ah, 01Ch až 01Fh a 080h až 09Fh.

Když n = 0, tak zase považuj všechny řídicí znaky za řídicí.

ESC 't' n

Když n = 0, opiš první polovinu generátoru znaků do druhé poloviny. Tím vznikne generátor znaků podobný generátoru podle zvyklostí EPSON - s jedním rozdílem: generátor EPSON obsahuje ve druhé polovině kurzívu (tj. šikmé písmo), které my neumíme tisknout; proto tam máme stejné znaky jako v první polovině generátoru.

Když n = 1, vrať do druhé poloviny generátoru znaky podle zvyklostí IBM.

ESC 'R' n

(podle zvyklostí EPSON)

Uprav generátor znaků pro tisk národní sady znaků podle hodnoty n:

n = 0	americká	(#$@[\]^`{|}~)
n = 1	francouzská	(#à°çS^`éùèÄ)
n = 2	německá	(#$§ÄÖÜ^`äöüß)
n = 3	anglická	(£$@[\]^`{|}~)
n = 4	dánská I	(#$@ÆØÅ^`æøå~)
n = 5	švédská	(#¤ÉÄÖÅÜéäöåü)
n = 6	italská	(#$@°\é^ùàòèì)
n = 7	španělská I	(₧$@¡Ñ¿^`"Ñ}~)
n = 8	japonská	(#$@[¥]^`{|}~)
n = 9	norská	(#¤ÉÆØÅÜéæøåü)
n = 10	dánská II	(#$ÉÆØÅÜéæøåü)
n = 11	španělská II	(#$á¡Ñ¿é`{Ñóú)
n = 12	latinskoamerická	(#$á¡Ñ¿éÜ{Ñóú)

n = 13 až 255 česká a slovenská v kódu KOI-8čs.

ESC ':' NUL NUL NUL

(podle zvyklostí EPSON)

Opiš ROM-generátor do RWM-generátoru.

Paměť tiskárny obsahuje dvě oblasti pro dva generátory znaků. První z nich pro tzv. ROM-generátor, jehož obsah lze sice různě přeskupovat (např. posloupností ESC 'R' nebo ESC 't'), ale není možné do něj vkládat zcela nové znaky. To umožňuje druhý generátor znaků, tzv. RWM-generátor. Po zapnutí tiskárny neobsahuje RWM-generátor nic smysluplného, natož pak nějakou sadu znaků. Posloupností ESC ':' způsobíme, že jeho obsah bude shodný s obsahem ROM-generátoru. Tím si RWM-generátor připravíme k použití a některé znaky v něm pak můžeme podle potřeby nebo rozmaru změnit např. posloupností ESC '&', která bude popsána dále.

ESC '%' n

Když n = 1, přepni na RWM-generátor
když n = 0, přepni na ROM-generátor.

ESC 'I' n
(podle zvyklostí IBM)
Když n = 0, přepni na ROM-generátor a tiskni ledabyle.
Když n = 2, přepni na ROM-generátor a tiskni pečlivě.
Když n = 4, přepni na RWM-generátor a tiskni ledabyle.
Když n = 6, přepni na RWM-generátor a tiskni pečlivě.
Tato řídicí posloupnost je zopakována v kapitole 3.3.10.

ESC '&' NUL c1 c2 n1 m1 m2...m11 n2 m12...
(podle zvyklostí EPSON)

Nahraď v RWM-generátoru znaky c1 až c2. Každý znak je popsán číslem n a jedenácticí množin m. Takových popisů znaků je v řídicí posloupnosti ESC '&' tolik, kolik znaků má tiskárna v generátoru nahradit, počínajíc znakem c1 a končíc znakem c2. Popisy znaků jsou uvedeny v pořadí podle jejich kódů.

Číslo n popisuje, zda má být znak tištěn ve vrchních osmi linkách řádky nebo o jednu linku níž (to odpovídá vrchním nebo spodním osmi jehlám v hlavě; devátá, nejspodnější nebo nejvrchnější jehla, vždy zahálí). Kromě toho číslo n udává, ve kterém sloupci znak začíná a ve kterém sloupci znak končí. Toho se využívá při proporcionálním tisku, kdy se nevyužité sloupce netisknou.

Jak získáme číslo n? Napřed si připravíme tři čísla, která posléze sečteme: první číslo bude 0 v případě, že se má tisknout do spodních osmi linek, nebo 128 (tj. 80h) v případě, že se má tisknout do vrchních osmi linek. Druhé číslo je šestnáctinásobkem čísla počátečního sloupce. Znak smí začínat ve sloupci 1 až 7 a tomu odpovídají čísla

16, 32, 48, 64, 80, 96 a 112,
tj. 10h, 20h, 30h, 40h, 50h, 60h a 70h.

Třetí číslo udává poslední sloupec v rozsahu 3 až 11. Každý znak musí být zobrazen aspoň třemi sloupci.

Například znak 'i' se tiskne do vrchních osmi linek, začíná ve sloupci 3 a končí ve sloupci 7. Tomu odpovídají tři čísla

128, 48 a 7,
tj. 80h, 30h a 7h.

Číslo n získáme součtem:

n = 128 + 48 + 7 = 183 nebo pohodlněji
n = 80h + 30h + 7h = 0B7h.

Autor doporučuje pilnému čtenáři, aby tento příklad vyřešil ještě ve dvojkové soustavě. Pak teprve vystoupí prostota této záležitosti na povrch v celé své nahotě:

```
  10000000
+ 00110000
± 00000111
= 10110111
```

Jedenáctice množin m popisuje jedenáct sloupců v okénku, do kterého se znak vytiskne (viz kapitolu 3.1). Pro nás jsou však významné jen sloupce

1, 3, 5, 7 a 9,

zobrazené množinami m1, m3, m5, m7, m9 (u druhého znaku m12, m14, m16, m18, m20 apod.). Na sloupce 2, 4, 6, 8, 10 a 11 se vůbec nebere ohled. Musí být uvedeny jen z důvodů slučitelnosti s tiskárnami, které mají podrobnější generátor znaků než naše tiskárna. Obsah množin m se tvoří stejně jako při grafickém tisku, to je popsáno u posloupnosti ESC 'K'.

Např. posloupnost ESC '&' NUL 'ĽĽ' 88h
[t5, t6, t7] [] [t4, t5, t6, t7, t8] []
[t2, t3, t4, t5, t6] []
[t4, t5, t6, t7, t8] [] [t5, t6, t7] [] []
vloží do RWM-generátoru znak ♥.

ESC '=' n1 n2 n3 c n4 n5 m1...m11 n6 n7 m12...
(podle zvyklostí IBM)

Nahraď v RWM-generátoru znaky počínajíc znakem c. Číslo

n1 + 256 * n2 udává,

kolik prvků ještě bude mít tato řídicí posloupnost. Číslo n3 je vždy 20. Čísla n5, n7, n9 atd. jsou vždy nuly. Čísla n4, n6, n8 atd. mají stejný význam jako čísla n v posloupnosti ESC '&'. Množiny m mají také stejný význam jako v posloupnosti ESC '&'.

ESC '~' n

Když n = 1, nahraď znak nula v generátoru znakem přeškrtnutá nula "Ø" (viz u kódu 01Fh v tabulce 3.1).

Když n = 0, nahraď znak nula v generátoru znakem neškrtaná nula "0" (viz u kódu 030h v tabulce 3.1).

3.3.8 Tabulace

VT

Posuň papír vpřed na nejbližší následující zarážku svislého tabulátoru.

Po zapnutí tiskárny síťovým vypínačem se všechny zarážky nastaví na nulu a to znamená, že znak VT způsobí posun papíru o 1 řádek stejně jako LF. Nastavení zarážek však lze změnit některou z dále uvedených řídicích posloupností, např. ESC B.

HT

Další znaky tiskni až od polohy nejbližší následující zarážky vodorovného tabulátoru.

Po zapnutí tiskárny síťovým vypínačem se zarážky nastaví na pozice 9., 17., 25. ... znaku v řádce, vždy po osmi znakových pozicích. Nastavení zarážek lze změnit některou z dále uvedených řídicích posloupností, např. ESC D.

ESC 'R'
(podle zvyklostí IBM)
Nastav zarážky svislého i vodorovného tabulátoru jako po zapnutí síťovým vypínačem.

ESC '/' n
Přepni svislou tabulaci na kanál n.
Svislý tabulátor má celkem 8 kanálů očíslovaných 0 až 7. Po zapnutí tiskárny síťovým vypínačem se samovolně vybere kanál 0. Může nás však potkat taková nepříjemnost, že budeme muset tisknout do několika různých druhů předtištěných tiskopisů. V každém druhu tiskopisu je třeba vyplňovat jiné řádky. Proto každému tiskopisu přiřadíme jeden kanál svislé tabulace a zarážky v každém kanálu nastavíme na požadované řádky pro ten který tiskopis. Zarážky v kanálech se nastavují řídicí posloupností ESC 'b', o které ještě bude řeč. Posloupností ESC '/' pak volíme kanál podle tiskopisu, který budeme vyplňovat.

ESC 'B' n1 n2 ... NUL
Nastav zarážky v kanálu 0 svislého tabulátoru na hodnoty n1, n2 atd. Hodnoty n udávají polohu zarážky v řádkách od začátku stránky (v závislosti na právě nastaveném řádkování). Zbývající zarážky se nastaví na nulu a způsobí posun papíru o jednu řádku. Čísla n1, n2, ... musí být uspořádána vzestupně, od nejmenšího k největšímu.

ESC 'b' n1 n2 n3 ... NUL
Nastav zarážky v kanálu n1 svislého tabulátoru na hodnoty n2, n3 atd. Čísla n2, n3, ... musí být uspořádána vzestupně.

ESC 'D' n1 n2 ... NUL
Nastav zarážky vodorovného tabulátoru na znakové pozice n1, n2 atd. Čísla n1, n2, ... musí být uspořádána vzestupně.

ESC '$' n1 n2
Další znaky tiskni od sloupce n1 + 256 * n2 (znaménko * označuje násobení). Jedná se o sloupce teček s roztečí 1/60 krát 25,4 mm, tedy o sloupce ledabylého tisku. Sloupce se počítají od právě nastaveného levého okraje. Pokud nastavovaná pozice přesahuje pravý okraj, neprovede se vůbec nic.

ESC '\' n1 n2
(podle zvyklostí EPSON)
Další znaky tiskni od sloupce vzdáleného o n1 + 256 * n2 od polohy, na kterou se právě má tisknout. Jedná se o sloupce s roztečí 1/120 krát 25,4 mm, tedy o sloupce pečlivého tisku. Pokud nastavovaná pozice přesahuje právě nastavený levý nebo pravý okraj, neprovede se nic.

Nyní se ještě vtírá otázka, jak vyjádřit záporný počet sloupců pomocí n1 a n2. Záporné číslo napřed přičteme k 65536, tím dostaneme číslo menší než 65536 a to už můžeme vyjádřit dvojicí n1 a n2 (jako zbytek po dělení 256 a podíl 256). Znalec jistě poznal kódování záporných čísel v dvojkovém doplňkovém kódu.

3.3.9 Okraje a zpracování textů

CR

Následující znaky tiskni od levého okraje řádky.

ESC '5'
(podle zvyklostí IBM)
Když n = 1, odřádkuj při každém CR.
Když n = 0, neřádkuj při CR.

Tento řídicí znak zapíná a vypíná samovolné vykonávání činnosti, kterou obvykle způsobuje řídicí znak LF (řádkování). Proto se takové obohacení činnosti pro znak CR někdy nazývá také automatické řádkování.

BS

Vytiskni řádku a vrať se o znak zpět.

To znamená, že následující znak se umístí na tutéž pozici, na kterou se vytiskl poslední znak, takže poslední vytištěný a následující znak se budou překrývat. Použití několika znaků BS bezprostředně za sebou způsobí, že se bude překrývat několik znaků v řádce, např.: ▪▪▪.

ESC SP n

Prostrkávej znaky mezerou o velikosti n krát 1/120 krát 25,4 mm. Tato mezera bude vložena mezi každé dva znaky, které se budou od nynějška tisknout.

Např. ESC SP 6 způsobí: o m o m o m o m o m.

ESC 'a' n
Když n = 0, zarovnávej řádky podle levého okraje.
Když n = 1, zarovnávej řádky podle pravého okraje.
Když n = 2, umísťuj řádky doprostřed mezi okraje.
Když n = 3, zarovnávej přeplněné řádky podle obou okrajů.

K zarovnávání podle obou okrajů je třeba připomenout, co se stane, když se znaky nevejdou do řádku. Najde se konec posledního dokončeného slova a tam se ukončí řádek. Pak se řádek zarovná. Po zapnutí tiskárny síťovým vypínačem se nastaví zarovnávání vlevo, ale to lze změnit právě posloupností ESC 'a'. Pokud bylo nastaveno zarovnávání podle obou okrajů, vyvstává před tiskárnou problém, jak to udělat. Ne každý řádek vzniklý přeplněním má délku přesně od kraje do kraje. Proto je třeba některé řádky prodloužit. To se udělá jednoduše tak, že chybějící délka se rozdělí do mezer mezi slovy. Mezery mezi slovy se zvětší a řádek se vytiskne přesně od levého do pravého okraje. Nyní je ještě třeba něco udělat se zbytkem řádku, který se nevytiskl. Jde jednak o mezeru mezi posledním dokončeným slovem a slovem, které přesáhlo pravý okraj (a dosud nebylo dokončeno), jednak o to nedokončené slovo. Tiskárna mezeru jednoduše vynechá a nedokončené (a dosud nevytištěné) slovo uloží do své paměti jako začátek následujícího řádku.

Předchozí odstavec je příkladem zarovnávání podle obou okrajů.

Zarovnání vlevo.

Středění.

Zarovnání vpravo.

ESC '1' n

Nastav levý okraj n mezer od nejzazšího možného levého okraje. Šířka mezery závisí na tom, jaký druh písma byl nastaven (o druzích písma pojednává následující kapitola). Po zapnutí tiskárny síťovým vypínačem má mezera šířku 12 sloupců.

Při nastavování okrajů se vytiskne řádka z paměti tiskárny, smažou se všechny zarážky vodorovného tabulátoru a následující znaky se pak budou tisknout od nového levého okraje.

Autor doporučuje laskavé pozornosti čtenáře skutečnost, že zmíněná řídicí posloupnost obsahuje malé písmeno 'l', nikoli snad číslici '1'. Příklad je uveden u posloupnosti ESC X.

ESC 'Q' n

(podle zvyklostí EPSON)

Nastav pravý okraj n mezer od nejzazšího možného levého okraje. Šířka mezery závisí na druhu písma stejně jako u předchozí posloupnosti.

Při nastavování okrajů se vytiskne řádka z paměti tiskárny, smažou se všechny zarážky vodorovného tabulátoru a následující znaky se pak budou tisknout od levého okraje.

Příklad je uveden u posloupnosti ESC X.

ESC 'X' n1 n2

Nastav levý i pravý okraj. Číslo n1 udává, na které znakové pozici se bude tisknout nejlevější znak v řádce (nejvíc vlevo je první pozice), n2 udává pozici znaku nejvíce vpravo na řádce. Šířka znakové pozice je obvykle šířka mezery. Řídicí posloupnost ESC 'X' se však liší od ESC 'l' a ESC 'Q' v případě, že se tiskne rošířené písmo (rozšířená mezera má dvojnásobnou šířku) - i v takovém případě je šířka pozice pouze jako jednoduchá mezera. U posloupností ESC 'l' a ESC 'Q' se ovšem počítá se šířkou rozšířené mezery.

Při nastavování okrajů se vytiskne řádka z paměti tiskárny, smažou se všechny zarážky vodorovného tabulátoru a následující znaky se pak budou tisknout od nového levého okraje.

Př.: Předchozí text byl tištěn s okraji nastavenými posloupností ESC 'X' 10 70. Nyní nastavíme ESC 'X' 48 57:

omomomomom
omomomomom
omomomomom
omomom

3.3.10 Písmo

ESC 'M'
Tiskni písmo élite. Toto je písmo élite.

ESC ':'
(podle zvyklostí IBM)
Tiskni písmo élite. Toto je písmo élite.

ESC 'P'
Tiskni písmo pica. Písmem pica je tištěn tento text.

ESC 'p' n
Když n = 1, tiskni proporcionálně.
Když n = 0, přestaň tisknout proporcionálně.

Tohle písmo není proporcionální.
Tohle písmo -je- proporcionální.

SI nebo **ESC SI**
Tiskni zhuštěné písmo. To je zhuštěné písmo.

DC2
Přestaň tisknout zhuštěně.

ESC 'E'
Tiskni tučně zdvojením sloupců. Zdvojení sloupců.

ESC 'F'
Přestaň tisknout tučné písmo zapnuté posloupností ESC 'E'.

ESC 'G'
Tiskni tučně zdvojením linek. **Zdvojení linek.**

ESC 'H'
Přestaň tisknout tučné písmo zapnuté posloupností ESC 'G'.

SO nebo **ESC SO**
Tiskni široké písmo do konce řádky (znak CR) nebo do vypnutí znakem DC4 nebo posloupností ESC 'W' nebo ESC '!'.
Široké písmo.

DC4
Přestaň tisknout široké písmo zapnuté znakem SO nebo ESC SO.

ESC 'W' n
Pro n = 1 tiskni široké písmo, pro n = 0 vypni široké písmo.

ESC '-' n
Pro n = 1 začni podtrhávat, pro n = 0 přestaň.

ESC '!' m
Zvol způsob tisku podle prvků množiny m. Neobsahuje-li m žádný prvek, nastaví se obyčejné písmo pica. Prvky množiny jsou příkazy pro nastavení neobyčejných druhů písma. Množina může mít tyto prvky:

tiskni písmo élite
tiskni proporcionálně
tiskni zhuštěné znaky
tiskni tučně zdvojením sloupců
tiskni tučně zdvojením linek
tiskni široké znaky
prázdný příkaz (u FX-85 tiskni kurzívou)
podtrhávej.

Dodejme, že když některý prvek v množině m není a přitom byl příslušný způsob tisku nastaven již dříve jinou řídicí posloupností, bude takový způsob tisku zrušen.

Pozorný čtenář si možná ještě pamatuje z kapitoly 2.4, že kvůli zobrazení v počítači musí být prvky množiny uspořádány. Poznamenejme, že prvky množiny m jsou uspořádány podle pořadí, jak jsou zde uvedeny.

ESC '_' n
Pro n = 1 začni nadtrhávat,
pro n = 0 přestaň.

Slovem nadtrhávat se rozumí vytvořit vodorovnou čáru nad znakem, kterou matematici obvykle nazývají slovem pruh.

Prostý lid však slovo pruh používá v jiných významech (kýla, jízdní pruh na vozovce apod.), zatímco slovo nadtrhávat používá prozatím jen jediný člověk na světě (Ing. R. Pecinovský, CSc.) právě v uvedeném významu (analogicky ke slovu podtrhávat).

ESC 'S' n

Pro n = 1 začni indexovat dole. To znamená, že následující znaky budou zmenšeny a vytištěny do spodních devíti linek v řádku, např.: spodní index

Pro n = 0 začni indexovat nahoře. To znamená, že následující znaky budou zmenšeny a vytištěny do vrchních devíti linek v řádku, např.: vrchní index.

ESC 'T'

Přestaň tisknout indexy.

ESC 'x' n

Pro n = 1 tiskni pečlivě, pro n = 0 ledabyle.

ESC 'I' n

Když n = 0, přepni na ROM-generátor a tiskni ledabyle.
Když n = 2, přepni na ROM-generátor a tiskni pečlivě.
Když n = 4, přepni na RWM-generátor a tiskni ledabyle.
Když n = 6, přepni na RWM-generátor a tiskni pečlivě.
Tato řídicí posloupnost již byla popsána i v kapitole 3.3.7.

3.3.11. Vynechané řídicí posloupnosti

Tiskárna EPSON FX-85 zpracovává navíc ještě tyto řídicí posloupnosti:

ESC '4' - tiskni kurzívou,
ESC '5' - přestaň tisknout kurzívou,
DC3 - přestaň tisknout přijímané znaky, dokud nepřijmeš DC1,
DC1 - začni opět tisknout,
ESC EM c - řízení podavače listů papíru.

Podavač listů papíru je zařízení, které vloží do tiskárny další list vždy, když se začíná tisknout nová stránka. To způsobí řídicí znak FF, kromě něj to však mohou způsobit i řídicí znaky nebo posloupnosti LF, VT a ESC 'J'.

Když c = '4', podavač se spřáhne s tiskárnou a začne ji obsluhovat podle jejích potřeb.

Když c = '0', podavač se vypřáhne a přestane si všímat tiskárny, byť by potřebovala další papír.

3.4. Podrobně o technických prostředcích

Tato kapitola je určena čtenářům, které zajímá podrobné řešení naší úlohy. Autor se obává, že tato kapitola může zahltit úplné začátečníky přemírou nových pojmů, které zatím není třeba znát. Z hlediska zadání a řešení úlohy však podrobný popis technických prostředků patří právě sem. Autor doporučuje, aby si čtenář tuto kapitolu zatím jen zběžně prolistoval, aniž by se snažil zapamatovat si nové pojmy a porozumět jim. Později bude tak jako tak nutné vracet se k podrobnostem uvedeným v této kapitole, takže pro začátek stačí vědět, kde se co najde. Výklad v této kapitole předpokládá znalost kapitoly 3.2.

Naše tiskárna je vlastně malým jednoúčelovým počítačem, který bude sloužit jinému počítači k výstupu informací na papír. Podobně jako velké počítače má i naše tiskárna procesor a paměť. Do pojmu paměť jsme zahrnuli i porty (to jsou ta zvláštní paměťová místa, která si nepamatují, co do nich procesor zapsal, nýbrž to sdělují nějakému výstupnímu zařízení a naopak zase předávají našemu procesoru informace z nějakého vstupního zařízení).

Nejjednodušším vstupním a výstupním zařízením, které tiskárna má, je konektor. Konektor je mnohakolíková zásuvka, do které se připojí kabel z nadřízeného počítače. O tomto spojení pojednává kapitola 3.4.2.

Dalším velmi jednoduchým vstupním zařízením je spínač. Sděluje přes port procesoru, zda je sepnut nebo rozepnut. Jeden spínač je umístěn na levé straně pod válcem tiskárny, a když hlava najede na levý kraj, tak ho sepne. Tento spínač říká procesoru, zda se hlava nachází či nenachází na levém kraji. Jiný spínač je zase stlačen papírem zasunutým do válce tiskárny. Když papír dojde, spínač se rozepne. Další dva spínače jsou v tlačítkách na vrchním panelu a čtyři přepínače uvnitř v tiskárně slouží jako předvolba. Podle jednoho přepínače předvolby si tiskárna připraví generátor znaků, druhý jí oznamuje, kolik měří stránka papíru (11 nebo 12 krát 25,4 mm), třetí jí říká, zda má přeskakovat perforaci mezi stránkami a čtvrtý oznamuje, zda má pracovat podle zvyklostí IBM nebo EPSON.

Aby mohla tiskárna pracovat, musí být motory napájeny správným napětím. To je potřeba zkontrolovat. Od napájecího napětí motorů je proto odvozen signál, který oznamuje, zda motory jsou nebo nejsou správně napájeny.

Nejen vstupní, ale i výstupní zařízení mohou být velmi jednoduchá. Mezi takhle jednoduchá výstupní zařízení patří dvě světýlka na vrchním panelu tiskárny. Když zapíšeme do příslušného bitu v portu hodnotu 1, světýlko se rozsvítí, když 0, tak zhasne.

Posledním jednoduchým výstupním zařízením je pípák, o němž jsme se zmínili v kapitole 3.2.

Již jsme podrobně probrali uspořádání tiskací hlavy i podstatu její činnosti. Nyní se ještě zamysleme nad tím, jak rychle budeme tisknout. Rychlost tisku je omezena:

- jednak hustotou tisku (udáváme, kolik sloupců chceme tisknout na 25,4 mm),
- jednak dobou potřebnou k vysunutí a zatažení jehel
- a nakonec také rychlostí pohybu hlavy.

Naším cílem bude tisknout co možná nejrychleji. Budeme muset vykonat několik pokusů, abychom našli potřebné hodnoty těchto tří omezujících veličin. Ale už teď musíme odhadnout hodnoty alespoň vyhovující, abychom vůbec mohli nějaké pokusy dělat.

Největší smysluplná hustota tisku je určena jednak tloušťkou jehly (a tím i velikostí tečky, kterou jehla otiskne na papír), jednak přesností, s jakou dokážeme tečku umístit. Tyto veličiny jsou dány konstrukcí mechanické části tiskárny, tedy celkovým záměrem zákazníka a technickými možnostmi konstruktéra. V našem případě má tečka na papíru průměr aspoň 0,25 mm a hlavu posuneme jedním krokem o 1/120 krát 25,4 mm. Přitom vůle v převodech vnášejí do nastavování polohy hlavy další nepřesnost, která běžně dosahuje velikosti 1 kroku. Proto nemá smysl, abychom se pokoušeli o větší hustotu než 120 sloupců na 25,4 mm.

Doba potřebná na vysunutí jehly byla konstruktérem elektrické části tiskárny nastavena na 0,7 až 1 ms a doba zatažení jehly zpět je ještě o něco delší, takže musíme počítat aspoň 2 ms na vytisknutí jednoho sloupce.

Rychlost pohybu hlavy můžeme ovládat ve velkém rozsahu pomocí krokového motoru a předpokládáme, že ji budeme ovlivňovat dokonce i řídicí posloupností ESC 's' a že se bude měnit i podle hustoty tisku. V kapitole 3.4.1 podrobně rozebereme různé možnosti ovládání krokových motorů a upozorníme na nástrahy, které jsou na nás nalíčeny. Dojdeme k závěru, že pro začátek vystačíme s rychlostí 1 krok za 3 ms.

Pokud jde o barvicí pásku, musíme zajistit, aby se pohybovala jinou rychlostí než hlava. Kdyby se totiž pohybovaly stejně rychle, bouchaly by jehly do stále stejného místa na pásce. Buď by se jehly zasekly do pásky a páska by pak odtáhla hlavu do nepředvídatelné polohy, nebo by páska po několika úderech přestala barvit, takže by se přestaly dělat černé tečky.

3.4.1. Pochodem vchod, zastavit stát!

Zabývejme se trochu podrobněji krokovými motory. Co musíme udělat, aby krokový motor udělal krok? Jak rychle se může krokový motor točit? - To jsou otázky, na něž se pokusíme odpovědět.

Každý motor se skládá ze dvou částí: jedna stojí a nazývá se stator a druhá se v té první otáčí a říká se jí rotor. Ve statoru jsou důmyslným způsobem navinuty dráty tak, aby tvořily smyčky, ve kterých pak vzniká magnetické pole. Krokový motor má zpravidla čtyři vinutí. Pustíme-li do některého vinutí proud, začne silou magnetického pole přitahovat rotor, který udělá krok. Vypneme-li proud z toho vinutí, které právě přitahuje rotor, a pustíme proud do sousedního vinutí, rotor se opět pootočí o krok. Proto musíme spínat vinutí postupně, buď v pořadí

1 2 3 4 1 2 3 4 1 2 3 4...,

nebo raději vždy dvě vinutí najednou:

12 23 34 41 12 23 34 41 12 23....

Kromě toho můžeme oba způsoby kombinovat:

1 12 2 23 3 34 4 41 1 12 2 23 3 34....

Jednoduchou úvahou může bystrý čtenář dojít k názoru, že třetí způsob má dvakrát jemnější krokování než první dva. S tím lze skutečně souhlasit a někdy toho lze i účelně využít. V našem případě je však třetí způsob nepoužitelný, protože kroky jsou méně rovnoměrné než u prvních dvou způsobů. Pro naši potřebu si vybereme druhý způsob ovládání krokových motorů.

A jak budeme řídit krokový motor programem? Například by bylo možné připojit jednotlivá vinutí k výstupnímu portu počítače. Každý bit výstupního portu by pak zapínal a vypínal proud v některém vinutí motoru. Posouváním obsahu portu kolem dokola bychom pak roztočili motor.

Nám však konstruktér zjednodušil práci. Mezi procesor a motor vložil obvod, který se nazývá čítač. Čítač je paměť na jedno číslo, které (jak napovídá název) se může zvětšovat nebo zmenšovat o jedničku. Stačí, když zapíšeme do určitého portu cokoli, tím se zvětší obsah čítače, který je připojen k procesoru místo portu. Zapíšeme-li do jiného určitého portu cokoli, obsah téhož čítače se zmenší. Hodnota obsažená v čítači se pak bez našeho přispění správně dekóduje na takovou kombinaci zapnutých a vypnutých vinutí, aby motor udělal jeden krok. Jestliže jsme obsah čítače zvětšili, udělá se krok dopředu. Jestliže jsme obsah čítače zmenšili, udělá se krok nazpátek.

Nyní ještě zbývá zodpovědět otázku, jak rychle můžeme motor roztočit. Rychlost otáčení motoru závisí na tom, jak často budeme zapisovat do portu, ke kterému je motor připojen. Například rychlost 100 kroků za sekundu jistě zvládne každý motor. Když budeme zkoušet vyšší rychlosti, zjistíme že od určité rychlosti (např. 1000 kroků za sekundu) se už motor nedokáže roztočit. To je tím, že od něj vyžadujeme další krok ještě dřív, než stačil dokončit předchozí. Rotor pak nestačí běhat od jednoho vinutí ke druhému, zastaví se, bezmocně sebou cuká a bezradně bzučí, protože neví, kam dřív skočit.

Pro počáteční pokusy s tiskárnou nám jistě bude stačit i malá rychlost tisku, a proto vystačíme se zmíněnou jednoduchou metodou řízení krokových motorů. Během pokusů si však ujasníme,

jaké rychlosti potřebujeme dosáhnout, a případně si i vyzkoušíme následující způsob, jak zvýšit rychlost krokových motorů.

Co tedy musíme udělat, aby se motor točil rychleji, než jak jej dokážeme rozběhnout? Především si musíme uvědomit důležitou věc: Vinutí, kterým protéká proud, působí silou na rotor. Tato síla musí pohnout hmotou rotoru, převodů a všeho ostatního, co se má hýbat. Kromě toho musí ještě překonat tření, tedy sílu, která brzdí pohyb rotoru. Síla a hmotnost - nic víc, nic míň.

Bystrému čtenáři by měly pojmy síla a hmotnost připomenout jeden z fyzikálních zákonů, na kterém je založeno evropské novověké myšlení a o který se opíraly představy lidstva o světě až do začátku tohoto století. Pak se sice objevil pan Einstein, který staré názory s ostatními lidmi nesdílel a výklad světa značně zatemnil teorií relativity. Většina lidí se však nenechala zmást a přidržela se starého jednoduchého výkladu, kterému může každý snadno porozumět a s nímž kdekdo vystačí po celý život. Mnohému středoškolsky vzdělanému čtenáři možná v této chvíli vstávají vlasy hrůzou na hlavě při vyslovení jména Isaac Newton a při vzpomínce na zkoušení ze zákona síly. Nuže čtenáři, odlož svůj děs a pomni, že pohyb hmotného tělesa se vlivem síly zrychluje nebo zpomaluje.

Tímto návratem do školních škamen chtěl autor pouze přivést čtenáře na myšlenku, že by snad bylo možné otáčení motoru postupně zrychlovat. Začneme u rychlosti, kterou se motor ještě bezpečně roztočí a budeme přidávat, až se dostaneme na požadovanou rychlost. Stejným způsobem pak motor zabrzdíme, aby nepřeskočil nějaké kroky a nepomátl se tím (nacházel by se pak v jiné poloze, než kterou předpokládáme, a pak bychom tiskli na jiné místo, než kam potřebujeme).

Nedůvěřivý čtenář již možná pochybuje o obecné platnosti toho, co zde autor tvrdí - a dobře činí! Nikdo si asi nedělá plané naděje, že by krokovým motorem mohl překročit rychlost světla, přestože to z našich úvah vyplývá (pokud je dotáhneme do krajnosti). Jsou okolnosti, na něž se zapomnělo, jak tvrdí Franz Kafka. A ještě ke všemu úmyslně, abychom si zjednodušili život. Konec konců každá teorie má meze své platnosti a je použitelná jen v určité omezené oblasti. A ty meze je dobré znát. Protože když je neznáme, tak to ještě neznamená, že nejsou. Zkrátka a dobře se potvrzuje, že praxe je kritériem pravdy (Marx). A že pravda dotažená ad absurdum není pravdivá, nýbrž absurdní (Lenin).

Nám se naštěstí nemůže stát nic horšího, než že se motor při našem zrychlování asi tak po pátém kroku zastaví. Při tření totiž vzniká síla, která se zvětšuje tak, jak stoupá rychlost. Proto nemůžeme zrychlovat pohyb motoru rovnoměrně, nýbrž čím dál tím míň a nakonec už vůbec ne. A tak je nejlepší zjistit doby kroků při rozběhu a brzdění motoru zkusmo.

Ještě připojme malou poznámku o brzdění motorů. Autor považoval při svých pokusech za rozumné prodloužit první krok při brzdění asi na dvojnásobek. Představuje si totiž, že rotor zůstává v závěsu za polohou, která je určena právě sepnutým vinutím. Prostě se nechá táhnout, jako by byl přivázán na gumičce nebo na pružině (a my ho musíme táhnout pomalu, aby se nám neutrhl). Když ale chceme brzdit, tak ho musíme táhnout nazpátek. Proto ho necháme předběhnout a počkáme, až zase pěkně napne pomyslnou gumičku. Potom ho dobrzdíme stejně, jako jsme ho rozbíhali, jenom doby kroků budou v opačném pořadí, aby se zpomaloval.

3.1.4.2. Jak se stroje spolu domlouvají

Ten dělá to a ten zas tohle a všichni dohromady udělají moc. Když si lidé rozdělí práci tak, aby každý dělal to, co umí lépe než ostatní, udělá se víc a líp, než když každý dělá všechno sám. Tomu se říká dělba práce. A když vezmeme v úvahu, že někdo tu práci musí rozdělovat a někdo ji musí taky vykonávat, že někdo má vlohy k estrádnímu a varietnímu umění, kdežto jiný k tragickému dramatu, někdo rád poroučí, další má sklony k mučednictví, jiný zase nechápe, proč by měl táhnout, když se veze, a nakonec se najde třeba i takový, kdo se nehodí vůbec k ničemu, tak tomu můžeme říkat dělba údělu jako kdysi Jan Werich. Není však důležité jen to, aby ten, kdo poroučí, měl komu poroučet, a ten, kdo poslouchá, měl koho poslouchat. Je taky důležité, aby ten, komu se poroučí, slyšel, a když už slyší, tak aby taky naslouchal. Zkrátka a dobře je důležité, aby se lidé mezi sebou dokázali domluvit, protože jinak by každý dělal to, co nikdo nechce, a nikdo by nesehnal nic z toho, co zrovna potřebuje.

A podobně je tomu i se stroji. Snad proto, že je sestrojili lidé k obrazu svému (a často víc, než si uvědomovali a chtěli). I naše tiskárna se potřebuje domluvit s nadřízeným počítačem, který jí poroučí, co má dělat. Tiskárna je pokorným a přičinlivým sluhou počítače. Když splní zadanou práci, oznámí mu, že může přijímat další úkoly. Může se však stát, že tiskárna nemůže práci vykonat třeba proto, že nemá papír. Pak by počítač marně čekal na splnění úkolu. Proto je třeba, aby si tiskárna uměla také postěžovat, když ji postihnou nějaké potíže.

Teď nám ještě zbývá zodpovědět dvě otázky: z čeho to udělat a jak to udělat? Na první otázku je odpověď snadná. Znají ji všichni muži, kteří musí doma pomáhat své ženě. Manželka ráno zvedne telefon a kabeluje svému muži do práce, co všechno je třeba nakoupit. Poté muž okamžitě odkládá svou lopatu, kladivo, tužku, klávesnici, láhev rumu nebo cokoliv jiného, co ho zrovna zaměstnává, hbitě vyřizuje všechny úkoly a opět pomocí kabelu

sděluje své majitelce, že vše splnil, zároveň se žádostí o povolení vycházky s kamarády do hospody na pivo. Podobně i tiskárna dostává příkazy od počítače po kabelu a týmž kabelem hlásí počítači svůj stav.

Zbývá tedy ještě otázka, jak je to uděláno. Tiskárna i počítač se musí domlouvat nějakou společnou řečí. Kabel musí mít smluvený počet vodičů a na každém vodiči se musí ve smluvených okamžicích objevovat smluvené informace. Musíme se tedy domluvit, jak bude vypadat

rozhraní

mezi počítačem a tiskárnou. Náš vzor, tiskárna EPSON FX-85 obvykle používá rozhraní CENTRONICS. Této normy se přidržíme i my.

Především se předpokládá, že tiskárna bude blízko u počítače, a proto se volí jednodušší způsob dorozumívání za cenu většího počtu vodičů v kabelu: každý vodič v každém okamžiku přenáší jednobitovou informaci a všecky potřebné informace se vždycky přenášejí najednou, tedy po několika vodičích souběžně. Znalci tomu říkají paralelní přenos.

Počítač i tiskárna musí předem vědět, na kterém vodiči najdou kterou informaci. Každý vodič v kabelu přenáší 1 bit - tedy informaci o pravdivosti nebo nepravdivosti určitého výroku, o členství nebo nečlenství nějakého prvku v množině nebo číslo 0 či 1. Přenos informací pomocí vodičů je vysvětlen v kapitole 2.3.

Kabel se připojuje do tiskárny konektorem typu Amphenol, který má 36 vývodů (pro 36 vodičů v kabelu). Vývody jsou očíslovány a signály na nich se označují anglickými zkratkami. Mají tento význam:

1. $\overline{\text{STROBE}}$. Puls do nízké úrovně v trvání aspoň půl mikrosekundy říká tiskárně, že má přijmout další údaj.

2 až 9. DATA 1 až DATA 8. Osm bitů údaje přenášeného z počítače do tiskárny. Vysoká úroveň má číselnou hodnotu 1 nebo hodnotu výroku "je prvkem množiny". Znaky jsou zobrazovány svými pořadovými čísly podle kódové tabulky (tabulka 3.1a).

10. $\overline{\text{ACKNLG}}$. Puls do nízké úrovně v trvání aspoň 12 mikrosekund oznamuje počítači, že tiskárna očekává další rozkazy.

11. BUSY. Vysoká úroveň oznamuje počítači, že tiskárna je zaměstnána tiskem, obsluhou vrchního panelu nebo chybou.

12. PE. Vysoká úroveň oznamuje počítači, že došel papír.

13. Tento vodič je v tiskárně připojen na napětí +5 V přes odpor 3,3 kiloohmu.

14. $\overline{\text{AUTOFEED-XT}}$. Tento signál není v naší tiskárně vůbec použit. U FX-85 přikazuje v nízké úrovni tiskárně, aby samovolně řádkovala při každém znaku CR (tj. automatický LF).

15. Nepoužito.

16. Napětí 0 voltů.

17. Vodič spojený s kostrou tiskárny.
18. Nepoužito.

19. až 30. Napětí 0 voltů.

31. $\overline{\text{INIT}}$. Nízká úroveň po dobu delší než 50 mikrosekund uvede tiskárnu do stavu jako po zapnutí síťovým vypínačem.

32. $\overline{\text{ERROR}}$. Tiskárna oznamuje počítači chybu nízkou úrovní.

33. Napětí 0 voltů.

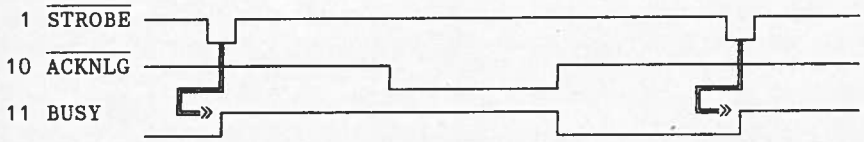
34. Nepoužito.

35. Viz vývod 13.

36. $\overline{\text{SLCT IN}}$. Nízkou úrovní počítač nařizuje tiskárně: spřáhni se se mnou! Vysoká úroveň umožňuje u FX-85 spřahování a vypřahování tiskárny řídicími znaky DC1 a DC3. Naše tiskárna si však nebude těchto řídicích znaků vůbec všímat.

Znavený spisovatel by nyní již rád odložil své pero, avšak zvídavý čtenář by mu v tomto okamžiku ještě určitě nedal spočinout, neboť mu jistě v hlavě hlodá pochybnost: a tohleto má stačit k tomu, aby se tiskárna domluvila s počítačem? Jak to domlouvání vlastně bude vypadat? Přidej, líný spisovateli, ještě aspoň popis časového průběhu uvedených signálů! Nuže, autor již spěchá, aby dal vše do pořádku.

Podívej se tedy, čtenáři, na tento obrázek:



Číslo na začátku řádky je číslem vývodu na konektoru, dvojitá lomená čára znázorňuje závislost signálů, jednoduchá lomená čára

znázorňuje vysokou a nízkou úroveň. Zleva doprava plyne čas, takže čáry zachycují úrovně signálů podobně jako film.

Nalevo jsou všechny signály v základním stavu, kdy se nic neděje: tiskárna nemá nic na práci a počítač nepotřebuje tisknout. Pak ale počítač potřebuje vytisknout nějaký znak. Podívá se na signál 11, kterým mu tiskárna sděluje, že nemá co na práci, vyšle znak po vodičích 2 až 9 a způsobí krátký puls signálu 1. Tiskárna uhlídá krátký puls na signálu 1 (viz šipku ⟹), zvedne signál 11 do vysoké úrovně (tím oznámí počítači, že je zaměstnána) a přijme a zpracuje znak. Když je hotova, oznámí to dlouhým pulsem signálu 10 a shodí signál 11 do nízké úrovně. Tím oznámí počítači, že už zase nemá co na práci. Tento postup se opakuje až do vypnutí tiskárny nebo počítače ze sítě.

3.4.3. Časovač typu 8253

Předchozí výklad jistě vyvolal v myslích přemýšlivých čtenářů příval otázek. Velkou skupinu z nich tvoří otázky spojené s časováním: Jak se pozná, že je na čase vykonat další krok motoru? Kdo určí, kdy se má bouchnout jehlami? Jak vyloudit z pípáku tón správné délky? Pilný student středoškolské fyziky zřejmě již tuší, že nejen délka tónu, nýbrž i výška tónu souvisí s časováním (takový čtenář může klidně přeskočit následující odstavec - až ke značce "***").

Krátký výlet do hudební akustiky

Někteří lidé se domnívají, že slyší zvuk, jakmile se jim rozechví v uchu blána zvaná bubínek. Bubínkem zpravidla chvěje vzduch, který přenáší třes nějakého předmětu (např. papírové blány reproduktoru). Vzduch tedy funguje jako nějaká vždy, všude a kdekým použitelná oj, která spojuje zdroj zvuku s uchem. (Představa, že by uši posluchačů v koncertní síni nebo na nějakém lidovém shromáždění mohly být spojeny se zdroji zvuku pomocí jiných než vzdušných ojí asi v praxi neobstojí.) Co je to třesení? Můžeme přijmout názor, že je to opakovaný pohyb sem tam.

Zůstává však otázkou, za jak dlouho se má jeden pohyb sem tam (neboli jeden kmit) vykonat: když totiž budeme třást pomalu, ozve se hluboké mručení. A když budeme třást rychle, vyloudíme vysoký pískot. A navíc všechny kmity určitého tónu musejí trvat stejně dlouho.

Melodie se skládá z tónů, které se od sebe liší přesným poměrem dob kmitů. Oktáva - to je polovina. U nejkrásnějšího tónu - komorního A - trvá 1 kmit 0,0022727 sekundy. Chceme-li vytvořit tón A o oktávu nižší, musíme prodloužit dobu kmitu na dvojnásobek: 0,0045454 s. Oktáva se skládá z 12 půltónů. V tzv. temperovaném ladění jsou všechny půltóny stejně velké. Proto jednomu půltónu odpovídá poměr dob kmitů rovný dvanácté odmocnině ze dvou, tj. 1,059463.

*Když tímto číslem vynásobíme dobu kmitu hlubokého tónu A, tedy 0,0045454 * 1,059463 = 0,0048157, dostaneme dobu kmitu tónu ais, po dalším násobení získáme tón h, pak postupně c, cis,d atd., až po dvanáctém násobení dostaneme tón o oktávu vyšší (tj. komorní A).*

(***)

Naším úkolem je roztřást blánu v pípáku. Reproduktor pípáku je elektroakustický měnič. Do pípáku tedy musíme přivést napětí, které se bude pohybovat sem tam. To bychom mohli udělat třeba tak, že budeme střídavě zapisovat nulový a jedničkový bit do nějakého portu. Abychom to však dělali správně často, abychom dodrželi zvolený kmitočet, k tomu potřebujeme něco, co by ve správnou chvíli oznámilo: už je na čase změnit hodnotu. To už je ale jednodušší, když toto upozornění "už je na čase" bude přivedeno přímo do pípáku, aniž by muselo být nějak zpracováno procesorem.

Všechny tři zmíněné úkoly spojené s časováním lehce splní součástka nazývaná časovač. Jedná se o integrovaný obvod typu 8253. Časovač obsahuje tři šestnáctibitové čítače (očíslované 0, 1 a 2) a řídicí obvod. Čítače i řídicí obvod jsou připojeny k portům procesoru. Procesor může zapsat do čítačů nějaké počáteční hodnoty. Každý čítač má svůj vstup a výstup. Na vstup čítače přicházejí pulsy. Každý puls způsobí, že se zmenší obsah čítače o jedničku. Obsah čítače se postupně zmenšuje až na hodnotu nula. Dočítání na nulu způsobí změnu napětí na výstupu. Co bude čítač dělat dál, to závisí na řídicím obvodu časovače, přesněji řečeno na tom, jak byl řídicí obvod naprogramován.

Čítač 0 je použit k časování programů. Dočítání na nulu je tedy významnou **událostí**, která musí být nějak sdělena procesoru a kterou musí program nějak ošetřit. To je zajištěno připojením výstupu čítače 0 na vývod procesoru NMI. O zpracování události NMI v procesoru se podrobně zmíníme později. Na vstup čítače 0 je připojen signál, který pulsuje každou mikrosekundu. Šestnáctibitový čítač dokáže čítat do 65536, takže je možné programovat události NMI až na 65 milisekund dopředu. To pro časování krokových motorů bohatě stačí.

Čítač 0 bude pracovat v módu 2 (to je třeba sdělit řídicímu obvodu na začátku programu). To znamená, že po dočítání na hodnotu 0 způsobí jednu událost NMI v procesoru a začne znovu čítat od původně zvolené hodnoty. Jestliže bude tato hodnota v průběhu čítání změněna, dokončí se čítání se starou počáteční hodnotou, dojde k události NMI a teprve pak se čítač nastaví na nově zvolenou počáteční hodnotu.

Čítač 1 řídí výšku tónu. Na jeho vstup je také připojen signál, který pulsuje každou mikrosekundu. Jeho výstup je veden do pípáku. Čítač pracuje v módu 3. To znamená, že po dočítání do nuly se na výstupu změní napětí a udrží se tam až do poloviny

následujícího čítání (tedy po polovinu naprogramované doby). Tím je zajištěno pravidelné chvění pípáku.

Čítač 2 řídí délku tónu. Na jeho vstup je připojen výstup z čítače 0. Chceme-li vyluzovat tón z pípáku, nastavíme nejdřív do čítače 0 hodnotu 1000, takže na vstup čítače 2 přijde puls každou milisekundu. Tím lze vytvářet tóny dlouhé od 1 ms až do 65 sekund. Čítač 2 pracuje v módu 0. To znamená, že po dočítání do nuly zablokuje výstup čítače 1 (takže signál výšky tónu se nedostane do pípáku, pípák se přestane třást a zmlkne).

Musíme se také zmínit o tom, jak sdělit časovači, ve kterých módech mají čítače pracovat. Do portu řídicího obvodu časovače postupně zapíšeme tři řídicí slova. Každé z těchto slov informuje časovač o tom, který čítač se má naprogramovat a ve kterém módu bude pracovat. Hodnoty řídicích slov, které použijeme, jsou uvedeny v kapitole 3.4.5.

Nyní ještě zbývá vysvětlit poslední maličkost: porty jsou jen osmibitové (řídicí slovo časovače je také osmibitové), ale čítače jsou šestnáctibitové. To znamená, že jejich obsah se musí zapisovat i číst nadvakrát: nejdříve osm méně významných bitů a potom osm významnějších bitů.

3.4.4. Vstupní a výstupní obvod typu 8255

Obvod 8255 je součástka, přes kterou jsou připojeny některé vstupní a výstupní signály. Je připojena ke čtyřem osmibitovým portům procesoru (podobně jako časovač). "Pětapadesátka" obsahuje tři bajty paměti (které mohou uchovávat vstupující nebo vystupující hodnoty). Tato paměťová místa jsou označena A, B a C. Kromě nich je v "pětapadesátce" také řídicí obvod, který jednak určuje, které z portů A, B a C budou sloužit pro vstup do procesoru a které k výstupu z procesoru, jednak umožňuje nastavovat jednotlivé bity ve výstupním portu C na jedničku nebo nulu, aniž by to mělo vliv na ostatní bity portu C.

Řídicím slovem 090h učiníme port A vstupním a porty B a C výstupními. Jak nastavovat bity na portu C pomocí řídicího bajtu? Bity v portu C očíslujeme od 0 do 7 (nejméně významný má číslo 0, nejvýznamnější 7). Toto číslo vynásobíme dvěma a přičteme k němu hodnotu 0 nebo 1 podle toho, co chceme vystoupit. Např. výstupu hodnoty 1 v nejvýznamnějším bitu na portu C dosáhneme, když do řídicího slova zapíšeme 2 * 7 + 1 = 14 + 1 = 15 (= 00Fh).

3.4.5 Noty

Podobně jako profesionální hudebník potřebuje i programátor nějaké noty, aby věděl, jak si má hrát se svěřeným strojem. Zadání úlohy si prostě musíme přehledně napsat. Vrchní rozhraní našeho programu, tedy požadavky na to, co má tiskárna umět, jsme již podrobně a přehledně popsali seznamem řídicích znaků a posloupností v kapitole 3.3.

Nyní ještě potřebujeme podrobný popis spodního rozhraní - rozhraní technických prostředků, se kterými chceme splnit svůj úkol. Tento popis musí obsahovat seznam všech portů, jejich adresy a význam každého bitu. U každého portu musí být napsáno, zda se z něj čte nebo zda se do něj zapisuje. Má-li bit hodnotu 1, pak je výrok uvedený v těchto notách pravdivý, má-li hodnotu 0, je výrok nepravdivý. Následující noty dodal autorovi konstruktér elektrické části tiskárny, učený kolega Matras.

Adresa 40h, zápis.
Na datech nezáleží.
Jeden krok hlavy doprava.

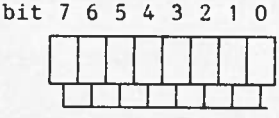
Adresa 41h, zápis.
Na datech nezáleží.
Jeden krok hlavy doleva.

Adresa 42h, zápis.
Na datech nezáleží.
Jeden krok papíru nahoru.

Adresa 43h, zápis.
Na datech nezáleží.
Jeden krok papíru dolů.

Adresa 44h, zápis.
Na datech nezáleží.
Bouchnutí jehlami, které byly nastaveny na portech 20h a 21h.

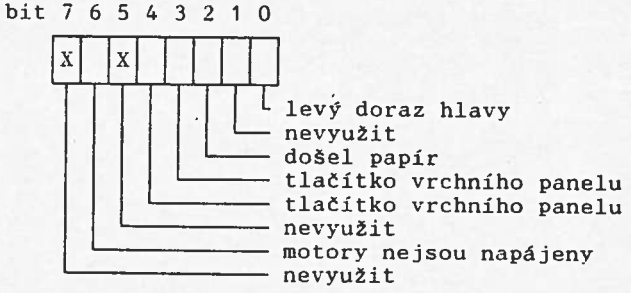
Adresa 00h, čtení.
Vstup dat přijatých z počítače (tj. vývody konektoru číslo 2 až 9).



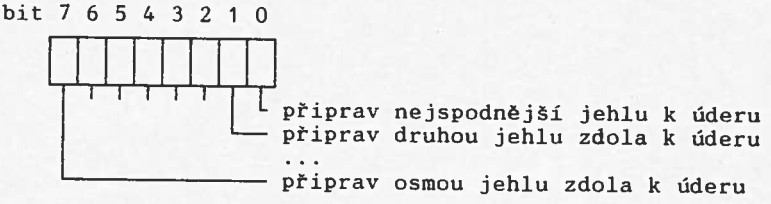
Vyslání dat z počítače do tiskárny je **událost**. O tom, že k takové události došlo, se musí procesor v tiskárně nějak dozvědět (aby mohl přijmout data). Proto zároveň s novými daty vzniká signál INT, který procesoru sděluje, že došlo k události "počítač posílá data". Je to podobná událost jako ta, která vzniká v časovači 8253 a projevuje se signálem NMI ("vypršel nastavený čas").

Adresa 10h, 11h, 12h, 13h, čtení i zápis.
Obvod 8253:
10h - čítač 0, čítá každou 1 mikrosekundu, po dočítání na hodnotu 0 způsobí přerušení NMI.
11h - čítač 1, výška tónu pro pípák.
12h - čítač 2, délka tónu pro pípák.
13h - řízení,
hodnota 034h - čítač 0 bude pracovat v módu 2
hodnota 076h - čítač 1 bude pracovat v módu 3
hodnota 0B0h - čítač 2 bude pracovat v módu 0.

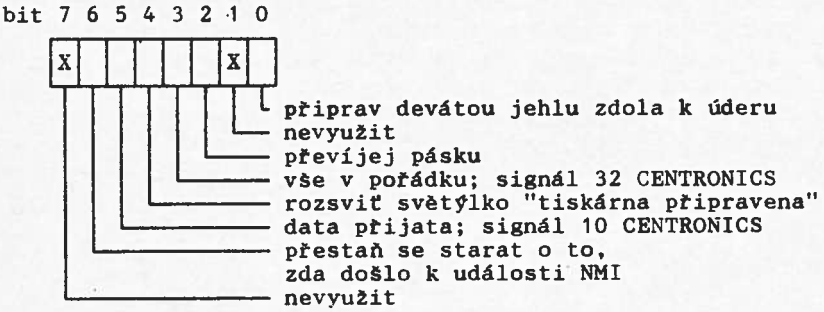
Adresa 20h, 21h, 22h, 23h, čtení i zápis.
Obvod 8255:20h - port A, čtení stavu tiskárny



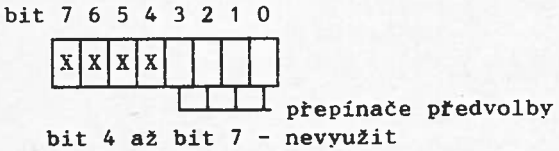
21h - port B, zápis, výběr jehel, které mají udeřit



22h - port C, zápis řídicích signálů



Adresa 30h, čtení



bit 4 až bit 7 - nevyužit

Shrnutí:

Kapitola podrobně popisuje vstupy a výstupy tiskárny, rozebírá princip činnosti a ovládání tiskací hlavy a krokových motorů.

4. Rozbor úlohy

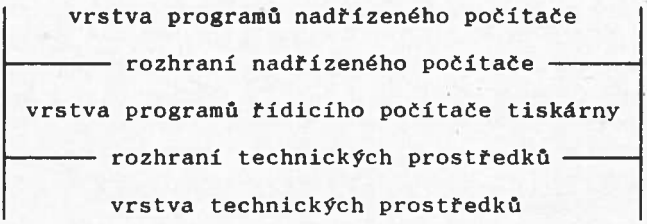
Rozbor je prvním krokem při programování úlohy. Proto je tato kapitola určena především programátorům. Kapitola však může být užitečná i zákazníkům, kteří si mohou vytvořit představu o tom, jak programátor pracuje se zadáním úlohy.

V minulé kapitole jsme vypracovali podrobné zadání úlohy. Určili jsme dvě rozhraní: rozhraní technických prostředků a rozhraní nadřízeného počítače, který posílá znaky do tiskárny (viz obr. 4.1). Naší úlohou je vytvořit vrstvu programů mezi těmito dvěma rozhraními.

Zmíněná dvě rozhraní oddělují naši vrstvu od dvou sousedních vrstev: Je to jednak vrstva technických prostředků, jednak vrstva programů nadřízeného počítače.

Pod rozhraním technických prostředků leží vrstva technických prostředků. Tato vrstva je tvořena prostředky, které jsme podrobně popsali v kapitole 3.4.

Na konec se ještě zmiňme o tom, co je nad rozhraním nadřízeného počítače. Pravděpodobně tam bude ovladač tiskárny, tedy systémový program, který zpřístupňuje tiskárnu uživatelům nadřízeného počítače. (Systémovým programem rozumíme součást operačního systému, tedy jisté vrstvy programů v nadřízeném počítači.)



Obr. 4.1

4.1. Jak na to

Naším úkolem je přeměňovat informace z rozhraní nadřízeného počítače na informace pro rozhraní technických prostředků. Z rozhraní nadřízeného počítače budeme přijímat znaky a ty máme přeměnit na různé signály s potřebnými časovými průběhy. Jak to uděláme?

Nejjednodušší by bylo najít takový procesor, kterému tuto činnost přikážeme jedinou instrukcí, a on ji pak bude vykonávat až do vypnutí ze sítě. Takový procesor však neexistuje, a to ze dvou důvodů: jednak by musel umět vykonat tak složitou instrukci, jednak by v něm musely být ukryty tak složité objekty, jako je

generátor znaků nebo paměť na řádku znaků. Instrukce běžných procesorů vykonávají jen velmi jednoduché činnosti, jako jsou množinové průniky a sjednocení, součty a rozdíly čísel. Obvyklé procesory pracují s nejjednoduššími objekty, aniž by rozlišovaly typ objektů. Procesor nezajímá, zda jsou v paměti uloženy množiny, celá čísla, znaky nebo logické hodnoty. Docela klidně sečte obsahy dvou buněk v paměti, z nichž jedna obsahuje třeba písmeno a druhá třeba logickou hodnotu. Obvyklé procesory však mají navíc jednu podstatnou vlastnost: dají se programovat.

Co je to programování? Programování umožňuje každý složitý objekt rozložit na jednodušší objekty a každou složitou činnost vysvětlit pomocí jednodušších činností jako pracovní postup. To není konec konců nic nového pod sluncem. Připomeňme třeba M.D. Rettigovou, která dokázala znamenitě naprogramovat několik generací českých hospodyněk k plné spokojenosti pánů manželů. Kdybychom se pokoušeli vysvětlit řešení celé úlohy v instrukcích procesoru, dozajista bychom utrpěli těžkou újmu na svém duševním zdraví. Je to asi stejně složité, jako kdyby např. ministr bezprostředně řídil činnost každého pracovníka ve svém resortu.

S touto obtíží se znamenitě vyrovnal Napoleon: sám velel pouze sedmi maršálům. Každý maršál velel sedmi generálům. A tak to šlo až k vojínům. Vojíni už neměli komu velet, tak do sebe nechávali střílet.

Vezměme si příklad z Napoleona a pokusme se vysvětlit naněkolikrát, jak a s jakými objekty má procesor úlohu vyřešit. To znamená, že vrstvu programů, kterou máme vytvořit, si uvnitř (a pouze pro vlastní potřebu) rozdělíme na jemnější vrstvy - tzv. technologické vrstvy. Každá nižší vrstva bude vysvětlovat činnosti a objekty použité ve vyšší vrstvě, až nakonec nejnižší vrstva bude napsána v instrukcích procesoru. Vzdělanci nazývají takovou soustavu hierarchický strukturovaný systém nebo inkrementální vrstvená architektura (vynikající kniha:

J. Voříšek: Architektura programového vybavení a řízení jeho tvorby, DT ČSVTS Praha, Praha 1987).

Pokud se však čtenář nechce stát vzdělancem, může si to pojmenovat, jak se mu zlíbí, neboť stojí psáno: "to, co růže nazývá se, zváno jinak, vonělo by stejně".

Pokusme se nyní navrhnout vrchní vrstvu našeho programu. Každá vrstva se skládá z jednoho nebo několika funkčních celků

modulů.

To, že dělíme vrstvu programů na funkční celky, má několik důvodů:

1. Když vytváříme modul, můžeme se zaměřit na jednu určitou funkci vrstvy a o ostatní funkce se nemusíme starat. Většinou se však nevyhneme tomu, aby se moduly navzájem neovlivňovaly.

Snažíme se tedy alespoň tyto vlivy co nejvíce omezit tím, že moduly mají co nejméně vazeb. To znamená, že používají co nejméně společných objektů a poskytují si navzájem co nejméně služeb.

2. Když program funguje chybně, snadno určíme, ve kterém modulu je chyba. Je to něco podobného, jako když třeba nestartuje auto. Ze struktury auta snadno určíme, ve kterém funkčním bloku máme hledat chybu: karoserie bude pravděpodobně v pořádku, chybu najdeme asi v motoru nebo v zapalování. Pokud se však řidič propadne podlahou na vozovku, nebude hledat chybu v motoru. Naproti tomu, když některý z diváků v divadle trefí herce shnilým rajčetem, budeme těžko zjišťovat, kdo to byl, protože dav v hledišti nemá potřebnou strukturu.

3. Když je třeba změnit některou funkci programu, stačí vyměnit jeden modul (nebo v horším případě několik málo modulů).

4. Programu, který je dobře rozčleněn do modulů, lze snadno porozumět, lze jej snadno pozměňovat, jeho funkci může pochopit i jiná osoba než autor.

5. Jednotlivé moduly mohou psát různí lidé. Jednak se tím zrychlí tvorba programu, jednak každý může programovat to, čemu rozumí.

6. Některé moduly lze opakovaně použít v několika různých programech.

7. Při zkoušení některých modulů lze ostatní moduly vynechat nebo aspoň zjednodušit. Záměnou modulů závislých na konkrétním počítači za jiné moduly lze programy přenášet na jiné počítače. Případně je možné simulovat chybějící technické vybavení tak, že místo modulů, které mají toto technické vybavení obsluhovat, se použijí moduly, které obsluhu technického vybavení pouze předstírají. Takže je možné současně vyvíjet program i technické vybavení. Program se přitom může vyvíjet na takovém počítači, který programátorovi dokáže poskytnout co nejlepší a nejpohodlnější prostředky pro tvorbu programu.

Na jaké moduly tedy rozdělíme vrchní vrstvu našeho programu? V podstatě nám jde o dva funkční bloky:
- modul ZZ, který bude zpracovávat znaky
  z nadřízeného počítače,
- modul RP, který bude ošetřovat vrchní panel tiskárny.

Věz, čtenáři, že jména modulů si autor vymyslel zcela svévolně (byť v souladu s pravidly, která předepisují obvyklé programovací jazyky pro tvoření jmen neboli identifikátorů). Obě jména jsou zkratky. Autor použil zkratek jednak proto, že se tato jména budou používat velmi často, jednak proto, že budou použita jako součást jiných, dlouhých jmen (tak aby se taková dlouhá jména ještě více neprodlužovala). Jinak ale bývá vhodné tvořit jména delší, aby byla srozumitelná. Jméno ZZ je zkratkou pro "zpracování znaků" a jméno RP je zkratkou pro "rozhraní panelu".

Modul ZZ bude vykonávat jedinou činnost - zpracovávat přijaté znaky. Ze zadání víme, ***co*** má procesor se znaky dělat. Nyní však budeme muset ještě vysvětlit, ***jak*** to má dělat. Napíšeme tedy pracovní postup, který bude pomocí nějakých jednodušších činností vysvětlovat, jak se mají znaky zpracovávat. Takový pracovní postup budeme nazývat

procedura.

Podobně jako ostatní objekty v modulu musí být i každá procedura napřed deklarována. Napíšeme-li pak do programu jméno procedury jako samostatný příkaz, tak procesor vykoná činnost předepsanou v proceduře. Takovýhle příkaz k vykonání procedury budeme nazývat zkráceně "příkaz procedury" nebo "volání procedury". Deklarace procedury vlastně způsobí, že k sadě příkazů programovacího jazyka přidáme další příkaz, který pak můžeme používat (stejně jako příkazy dané jazykem). Touto cestou můžeme přizpůsobit univerzální programovací jazyk své úloze tak, abychom ji mohli snadno a srozumitelně vyřešit.

Činnost zpracování znaků bude vykonávat procedura, kterou pojmenujeme Zpracuj_Znak. Jméno si opět vymyslel autor. Za povšimnutí stojí, kterak spojil dvě slova do jediného jména.

Modul RP bude vykonávat dvě činnosti:

1. Při zapnutí tiskárny musí nastavit volbu pečlivého nebo ledabylého tisku podle toho, zda je nebo není stisknuto tlačítko "POSUN" na vrchním panelu. Tuto činnost bude vykonávat procedura Nastav.
2. Musí průběžně sledovat tlačítko "RUČNÍ" na vrchním panelu a v případě, že shledá toto tlačítko stlačené, musí řídit činnost tiskárny v ručním režimu. Tuto činnost bude vykonávat procedura Ošetřuj_Panel.

Shrnutí:

Vysvětlili jsme pojmy vrstvená architektura, modul a procedura. Rozložili jsme vrchní vrstvu našeho programu na moduly ZZ a RP.

4.2. Hrubý návrh modulů ZZ a RP

Nyní se již můžeme pokusit o předběžný hrubý návrh modulů ZZ a RP. Oba moduly zapíšeme ve zjednodušené češtině (ve formě blízké programovacím jazykům). Autor považuje za nutné upozornit čtenáře, že se zatím nejedná o konečnou podobu modulů: jsou vynechány všechny podrobnosti a teprve až je postupně doplníme, získáme programy vhodné pro počítač.

Pisatel prosí čtenáře, aby si následující programy napoprvé jen zběžně prohlédl. Za programy následuje vysvětlení, které by

teprve mělo umožnit, aby čtenář vše náležitě pochopil.

```
Modul ZZ;
{ Zpracování znaků }
{ Napsal Ivan Ryant, 04.08.88/17:13 }

Vývoz
  Zpracuj_Znak;

Dovoz
  RZ, RVP, GZ;

{************************************************************}

Procedura Zpracuj_Znak (Předvolba: HW.Předvolba);
Proměnná
  Přečtený_Znak: bajt;

{============================================================}

Procedura Čti_Znak;
Začátek
  { Přečte znak z rozhraní znaků RZ a    }
  { uloží jej do proměnné Přečtený_Znak }
Konec {Čti_Znak};

{************************************************************}

Funkce Řídicí (x: bajt): logický;
Začátek
  { Zjisti, zda znak s kódem x je řídicí.          }
  { Když ano, vrať hodnotu pravda, jinak nepravda. }
  { Výjimka: pro znak DEL vrať hodnotu nepravda.   }
Konec {Řídicí};

{************************************************************}

Začátek {Zpracuj_Znak}
 { Počáteční nastavení proměnných }

 { Opakované zpracování znaku }
  Odtud_opakuj
    Čti_Znak;
    Platí_li Řídicí (Přečtený_Znak) tak
      { Vykonej činnost požadovanou řídicím znakem }
    jinak_platí_li Přečtený_Znak = 07Fh {tj. DEL} tak
      RVP.Smaž_Znak;
    jinak { tištitelný znak }
      GZ.Zpracuj_Znak (Přečtený_Znak);
```

```
    Konec {Platí_li};
  až_bude_platit nepravda;
Konec {Zpracuj_Znak};

{***********************************************************}

Konec { ZZ - zpracování znaků }.

Modul RP;
{ RP - Rozhraní vrchního panelu }
{ Napsal Ivan Ryant, 04.08.88/16:05 }

Vývoz
  Nastav, Ošetřuj_Panel;

{***********************************************************}

Funkce Nastav: logický;
{ počáteční nastavení - vrací, zda se má tisknout pečlivě }
Začátek
  { Napřed nastav počáteční hodnoty do proměnných      }
  {                                                    }
  { Pak se podívej, zda je stisknuto tlačítko "POSUN". }
  { Když je stisknuto, vrať hodnotu pravda,            }
  { jinak vrať hodnotu nepravda.                       }
Konec {Nastav};

{***********************************************************}

Procedura Ošetři;
{ ošetřuje vrchní panel, předpokládá pravidelné volání }
Začátek
  { Podívej se, zda je stisknuto tlačítko "RUČNÍ" nebo    }
  { zda chybí papír. Jestliže k ničemu takovému nedošlo,  }
  { nic nedělej.                                          }
  { Jestliže došlo, tak opakovaně sleduj tlačítka "POSUN" }
  { a "RUČNÍ". Je-li stisknuto tlačítko "POSUN", posuň    }
  { papír. Je-li stisknuto tlačítko "RUČNÍ", ukonči       }
  { činnost této procedury.                               }
Konec {Ošetři};

{***********************************************************}
Procedura Ošetřuj_Panel;
Začátek
  Odtud_opakuj
    Ošetři;
  až_bude_platit nepravda;
Konec {Ošetřuj_Panel};
```

{**}

Konec { RP - rozhraní vrchního panelu }.

Podívejme se na předchozí programy: na první pohled vidíme, že se skládají z různých slov a značek.

Tučná slova (jako **Modul, Začátek, Platí-li**) a různá znaménka (jako čárky, středníky, závorky, tečky, dvojtečky) mají přesně určený význam: označují příkazy a popisy (neboli deklarace). Příkazy říkají procesoru, co má dělat. Deklarace říkají procesoru, s čím má pracovat. Prvním slovem v každém modulu je slovo **Modul**, které říká: "Tady začíná deklarace modulu." Posledním slovem modulu je slovo **Konec**, které říká: "Tady končí deklarace modulu." Za slovem **Modul** následuje jméno modulu (ZZ, RP). Za jménem modulu je středník.

Pak následuje slovo **Vývoz** a seznam všech jmen konstant, typů, proměnných a procedur, které jsou v modulu deklarovány a jsou nabízeny k používání i ostatním modulům. Zbývající objekty (které sice jsou deklarovány uvnitř modulu, ale nejsou uvedeny v seznamu **Vývoz**) jsou soukromými objekty modulu a ostatním modulům bude jejich existence utajena. Seznam **Vývoz** je ukončen středníkem.

Poznamenejme, že jména konstant jsou taková jména, kterými vyjadřujeme nějakou pevně danou hodnotu: např. jmény pravda a nepravda vyjadřujeme hodnoty typu logický. Kromě takovýchto konstant, které jsou dány deklarací typu, můžeme pojmenovávat již známé konstanty, např.: Pí = 3,141592. Nebo třeba adresu portu, ke kterému jsou připojeny přepínače předvolby, můžeme pojmenovat: Přepínače_Předvolby = 30h. Takto utvořená jména konstant pak můžeme vyvážet z modulů, ve kterých byla deklarována tím, že je uvedeme v seznamu vyvážených objektů. Tak umožníme ostatním modulům používat tyto konstanty.

Za seznamem vyvážených objektů následuje podobný seznam - **Dovoz**. Tento seznam obsahuje jména těch modulů, jejichž vyvážené objekty chceme v našem modulu používat. Seznam je opět ukončen středníkem.

Středník je znaménko, které ukončuje každou deklaraci a každý příkaz uvnitř modulu. Jedině na konci celého modulu není středník - je tam tečka. Středníku je mezi programátory přisuzována nadpřirozená moc: mnozí věří, že programovací jazyk se středníky zaručuje srozumitelnost programu a zřetelnost a čistotu struktury programu. Autor však zvedá varovně ukazovák a doporučuje, aby si každý napřed dobře rozmyslel, co a jak chce programovat. Protože těm, kteří pálí své programy od boku a bez přemýšlení, kteří programují z jedné vody načisto a jsou ochotni kdykoli cokoli kamkoli přilepit, těm nepomůže ani kouzelný středník. Těm totiž nepomůže vůbec nic.

Dalšími důležitými značkami jsou složené závorky { a }. Mezi nimi se uvádějí komentáře. Komentář je libovolný text, o který

se počítač vůbec nezajímá. Tento text totiž slouží výlučně programátorovi. Je dobrým zvykem komentovat záhlaví každého modulu a každé procedury vysvětlením, co modul nebo procedura dělá. Také řádky plné hvězdiček jsou komentáře. V našich návrzích modulů ZZ a RP jsme použili komentáře také k tomu, abychom si poznamenali, jak budeme programovat jednotlivé procedury. Zatím nám totiž šlo jen o to, abychom si ujasnili, jaké procedury budeme potřebovat a co mají dělat. Příkazy do těchto procedur doplníme později.

Všechna ostatní slova, která nejsou ani tučná, ani nejsou uvedena v komentářích, jsou jména. Jedná se o jména konstantních hodnot (pravda, nepravda), typů (bajt, logický), proměnných (Přečtený_Znak), procedur (Zpracuj_Znak, Čti_Znak) nebo modulů (RVP, ZZ). Kromě toho se v programu vyskytují také desítková nebo šestnáctková čísla (07Fh).

Zásada je, že každé jméno v modulu musí být napřed nadeklarováno a teprve potom může být použito. To je také důvod, proč se programy píší a čtou odzadu. Když totiž píšeme program, zpravidla nevíme předem, jaké konstanty, typy, proměnné a procedury budeme potřebovat. Proto se nejprve podívejme na místo v proceduře Zpracuj_Znak uvedené komentářem "{ Opakované zpracování znaku }". Za tímto komentářem najdeme příkaz "**Odtud_opakuj...až_bude_platit** nepravda". Tři tečky nahrazují část programu, kterou bude procesor do nekonečna opakovat (protože nepravda nikdy platit nebude). A to je přesně to, co od procedury Zpracuj_Znak požadujeme: přijmout znak z nadřízeného počítače, pro řídicí znak vykonat jakousi činnost, znak, který není řídicí, vytisknout - a to celé opakovat až do vypnutí tiskárny ze sítě.

S přijetím znaku jsme se vypořádali velmi jednoduše: nadeklarovali jsme proměnnou Přečtený_Znak a proceduru Čti_Znak. Příkaz Čti_Znak způsobí, že se vykoná procedura Čti_Znak, která nějakým (prozatím blíže neurčeným) způsobem přijme znak z nadřízeného počítače a tento znak uloží do proměnné Přečtený_Znak. Snad se čtenář podiví, že proměnná Přečtený_Znak není typu znak, nýbrž bajt. Je tomu tak proto, že nejvíce se budeme věnovat netištitelným znakům, které je třeba označovat jejich číselným kódem nebo symbolickou zkratkou (LF, VT, SI apod.). Deklaraci proměnné Přečtený_Znak lze nalézt za záhlavím procedury Zpracuj_Znak. Za dvojtečkou je uveden typ deklarované proměnné.

Proč je deklarace proměnné Zpracuj_Znak uvedena mezi záhlavím procedury Zpracuj_Znak a začátkem této procedury? Proč není ještě před záhlavím procedury? To, že je uvnitř v proceduře, znamená, že její deklarace platí také jenom uvnitř v proceduře, a proto lze tuto proměnnou používat také jenom uvnitř procedury. Kdyby byla vně procedury Zpracuj_Znak deklarovaná ještě nějaká jiná procedura, nemohla by používat proměnnou Přečtený_Znak.

Podobně jako je proměnná Přečtený_Znak soukromou proměnnou procedury Zpracuj_Znak (a její jméno má jen místní platnost), je

i procedura Čti_Znak soukromou procedurou procedury Zpracuj_Znak. Takže proceduru Čti_Znak může používat jak procedura Zpracuj_Znak, tak i funkce Řídicí. Poznamenejme, že procedura Čti_Znak však nesmí používat funkci Řídicí, protože Čti_Znak je deklarována dřív než Řídicí (a žádné jméno nesmí být použito, dokud není deklarováno).

Nyní autor dluží čtenářům vysvětlení pojmu funkce. Funkce je zvláštní druh procedury. Podobně jako obyčejná procedura vykonává i funkce předepsanou činnost. Úkolem funkce však navíc je, aby zjistila nějakou hodnotu. Například funkce Řídicí má za úkol zjistit logickou hodnotu (tj. pravda nebo nepravda) výroku: "Znak x je řídicí znak".

Zatímco požadavek na vykonání obyčejné procedury zapíšeme samostatným příkazem, funkci můžeme použít jen tam, kde potřebujeme použít hodnotu. Jména funkcí používáme i jako součást složitějších výrazů. Např.: výraz Řídicí (Přečtený_Znak) **nebo** Řídicí (Přečtený_Znak - 080h) zjistí, zda je pravda, že hodnota uložená v proměnné Přečtený_Znak (případně zmenšená o hodnotu 080h) je kódem řídicího znaku. Takové a podobné výrazy (jejich výsledkem je vždycky nějaká hodnota) se používají jako součást příkazů.

Co se vlastně stane, když se vykonává příkaz procedury (nebo výraz obsahující volání funkce)? Až do okamžiku zavolání procedury řídil činnost procesoru volající program. A vyvolání procedury neznamená nic jiného, než že volající program zapůjčí proceduře procesor, aby procedura mohla být vykonána. Když procedura dokončí svoji činnost, vrátí procesor zpět volajícímu programu, který pak pokračuje dalším příkazem.

Jméno x v záhlaví funkce Řídicí je jméno tzv. parametru procedury. Parametr je taková zvláštní proměnná, do které se ukládá hodnota při vyvolání funkce (nebo procedury). Při vyvolání funkce Řídicí se do parametru x uloží hodnota obsažená v proměnné Přečtený_Znak (viz příkaz **Platí-li** Řídicí (Přečtený_Znak) **tak**...). O takto zadané hodnotě pak funkce Řídicí zjišťuje, zda je nebo není kódem řídicího znaku.

Vraťme se nyní k našemu programu. Uvnitř příkazu "**Odtud_opakuj...až_bude_platit** nepravda" jsou vloženy dva příkazy: první z nich vede k vykonání procedury Čti_Znak a druhý vykoná nějakou činnost s přečteným znakem podle toho, zda se jedná o znak řídicí (s výjimkou DEL), o znak DEL nebo o tištitelný znak. Druhý příkaz má tvar

```
Platí_li <logický výraz> tak
   <příkaz>
jinak_platí_li <logický výraz> tak
   <příkaz>
jinak
   <příkaz>
konec {Platí-li}.
```

Vidíme, že v tomto příkazu jsou vloženy další tři příkazy; každý z nich se však vykoná jen za určitých podmínek: první příkaz se vykoná, je-li Přečtený_Znak řídicí (ale není to DEL); druhý příkaz se vykoná, když Přečtený_Znak je DEL; třetí příkaz se vykoná, když Přečtený_Znak se má vytisknout.

V modulu ZZ se projevila potřeba pracovat se třemi novými funkčními celky, pro které později vytvoříme tři samostatné moduly.

Prvním z těchto funkčních celků je rozhraní nadřízeného počítače neboli rozhraní znaků. Rozhraní znaků bude ošetřovat modul RZ. A služeb tohoto modulu bude mj. využívat i procedura Čti_Znak.

Při zpracování znaku DEL potřebujeme odstranit poslední znak z řádku (uloženého ve vyrovnávací paměti řádku). Tato vyrovnávací paměť a práce s ní bude dost složitá na to, aby si zasloužila vytvoření samostatného modulu RVP. Modul RVP bude mj. vyvážet proceduru Smaž_Znak. Chceme-li tuto dováženou proceduru použít, musíme napřed uvést jméno modulu (a jméno procedury od něj oddělíme tečkou), protože stejné jméno (ale s jiným významem) může být dováženo i z jiných modulů, případně může být deklarováno v našem modulu.

Další poměrně dost složitou činností je vytištění znaku. Napřed je třeba najít v generátoru znaků správné kombinace teček a ty potom zobrazit na papír. Generátor znaků a operace s ním seskupíme do modulu GZ. Modul GZ bude mj. vyvážet proceduru Zpracuj_Znak (X: bajt) - s jedním parametrem nazvaným X, který je typu bajt.

Když si nyní čtenář prohlédne modul RP, jistě mu bude vše jasné - po formální stránce se jedná jen o jednodušší obdobu modulu ZZ a smysl je zřejmý na první pohled. Jedna věc však možná pozorného čtenáře zarazí: oba moduly (ZZ i RP) se tváří, jako by měl každý z nich svůj procesor. Jenže my víme, že oba budou zpracovávány jediným procesorem! Co si s tím jen počneme? O tom pojednáme v následující kapitole.

Shrnutí:

Navrhli jsme procedury obsažené v modulech ZZ a RP. Stanovili jsme, že nižší vrstvy našeho programu budou obsahovat moduly RZ, RVP a GZ.

4.3. Nekrájejte procesor!

Prchlivý čtenář se pravděpodobně chystá, že v příštím okamžiku odhodí tento spisek mezi odpadky. Snad se cítí oklamán a podveden schizofrenickými bludy pomateného spisovatele. Leč poshov, ukvapený čtenáři, neb spisovatel není pomaten do té míry,

jak se domníváš, ani z tebe nečiní oběť bohapustého žertování, když požaduje od dvou modulů, aby pracovaly současně na jediném procesoru!

Skutečně jde o to, aby modul RP sledoval vrchní panel současně se zpracováním znaků v modulu ZZ. To ovšem ještě neznamená, že rozkrájíme procesor na dva poloviční. Řešení nám nabízí kdysi běžná praxe výchovy dětí. V dávných dobách bývalo i u nás zvykem, že se děti staraly o své staré rodiče. Čím více měl rodič dětí, tím lépe o něj bývalo postaráno ve stáří. Dnes ovšem všichni víme, že rodič je lacinym zdrojem pracovní síly, peněz a nemovitostí a že vypotřebované rodiče lze (byť s jistými obtížemi) odložit do domova důchodců. Rodiče se tedy snaží mít co nejméně dětí, aby co nejdéle vydrželi v co nejlepším stavu.

Kdysi bývalo zvykem co rok, to nové dítě. Děti je ovšem třeba vychovávat po dobu několika let (nemluvě o těch, které se nepodaří vychovat do nejdelšího stáří). Jak to udělat?

Dítě je předmětem výchovného procesu. Tento proces není ukončen ani tehdy, když zrovna dítě není vychováváno - dítě prostě musí na výchovu počkat. Prostředkem, který musí být dítěti přidělen, aby mohl probíhat výchovný proces, je rodič. Výchovné procesy všech dětí probíhají současně, rodičů - prostředků je však omezený počet. Rodiče tedy musejí být sdíleni vychovávanými dětmi. Každý proces musí po konečné čekací době obdržet prostředek, po konečnou dobu může prostředek používat a pak jej musí předat dalšímu procesu.

V našem případě bude jeden proces probíhat na modulu ZZ a druhý proces na modulu RP. Sdíleným prostředkem je procesor. Vidíme, že souběžnou činnost dvou procesů na jednom procesoru lze uskutečnit. Jenom zůstává otázkou, jak a kdy si budou procesy předávat procesor. Odpověď na tuto otázku však vyžaduje ještě podrobnější rozbor úlohy. Proto zatím ponechme tuto otázku otevřenou s tím, že se k ní později znovu vrátíme.

Zatím zcela postačí, když si laskavý čtenář vytvoří představu o významu pojmů

proces

a

sdílený prostředek.

Nyní se pokusíme podrobně naprogramovat procedury Zpracuj_Znak, Nastav a Ošetři z modulů ZZ a RP. Vysvětlení opět následuje až za ukázkami programů. A teď už dost řečí, vrhněme se střemhlav do víru programování.

```
Modul ZZ;
{ Zpracování znaků }
{ Ivan Ryant, 25.06.88/17:13 }
```

```
Vývoz
  Zpracuj_Znak;

Dovoz
  RZ, HW, RU, OU, RVP, GZ;

Proměnná

 { Různé příznaky }
  Nastav_Bit_8,              { maska pro 7 bitový příjem }
  Zruš_Bit_8: bajt;          { maska pro 7 bitový příjem }
  Auto_LF,                   { samovolné řádkování při CR }
  Řídicí_Taky_Přes_80h,      { zapnuta druhá sada říd.Znaků }
  NUL_CAN_Řídicí: logický; { kódy 0..01Fh se netisknou }

{*********************************************************}

Procedura Zpracuj_Znak (Předvolba: HW.Předvolba);
Proměnná
  Přečtený_Znak: bajt;

{=========================================================}

Procedura Čti_Znak;
{ Naplní proměnnou Přečtený_Znak z RZ a ošetří 8. bit }
Začátek
  Přečtený_Znak := RZ.Vyjmi_Z_VP;
  Přečtený_Znak Typ HW.Osmice_Signálů :=
    (Přečtený_Znak Typ HW.Osmice_Signálů +
    Nastav_Bit_8 Typ HW.Osmice_Signálů) *
    Zruš_Bit_8 Typ HW.Osmice_Signálů;
Konec {Čti_Znak};

{*********************************************************}

Funkce Řídicí (x: bajt): logický;
Začátek
  Platí_li Řídicí_Taky_Přes_80h tak
    x := x mod 80h;
  Konec {Platí_li};
  Řídicí := (x < 020h) a_zároveň (NUL_CAN_Řídicí nebo
    ((x <> 000h {NUL}) a_zároveň (x <> 018h {CAN})));
Konec {Řídicí};

{*********************************************************}

Procedura Zpracuj_ESC;Začátek
  { Přečti řídicí posloupnost     }
  { a vykonej příslušnou činnost. }
```

```
Konec {Zpracuj_ESC};

{************************************************************}

Začátek {Zpracuj_Znak}
 { Počáteční nastavení proměnných }
  Nastav_Bit_8 := 0;
  Zruš_Bit_8 := 0FFh;
  Auto_LF := nepravda;
  Řídicí_Taky_Přes_80h := pravda;
  NUL_CAN_Řídicí := pravda;

 { Cyklus na zpracování znaku }
  Odtud_opakuj
   { Zpracuj znak z rozhraní znaků }
    Čti_Znak;
    Platí_li Řídicí (Přečtený_Znak) tak
      Vyber Přečtený_Znak mod 080h ze
      000h: { NUL }
        Začátek
          { nic nedělej };
        Konec;
      007h: { BEL }
        Začátek
          RVP.Tiskni_Kus_Řádku;
          RU.Přidej_Do_VP (OU.Zvonek);
        Konec;
      008h: { BS }
        Začátek
          RVP.Tiskni_Kus_Řádku;
          RVP.Smaž_Znak;
        Konec;
      009h: { TAB - zpracovává se jako tištitelný znak }
        Začátek
          GZ.Zpracuj_Znak (9);
        Konec;
      00Ah: { LF }
        Začátek
          RVP.Tiskni_Kus_Řádku;
          OU.Řádkuj;
        Konec;
      00Bh: { VT }
        Začátek
          RVP.Tiskni_Kus_Řádku;
          OU.Tabuluj_Svisle;
        Konec;
      00Ch: { FF }
        Začátek
          RVP.Tiskni_Kus_Řádku;
```

```
      OU.Stránkuj;
    Konec;
  00Dh: { CR }
    Začátek
      RVP.Tiskni_A_Doleva;
      GZ.Široké_Do_Konce_Řádku := nepravda;
      Platí_li Auto_LF tak
        OU.Řádkuj;
      Konec {Platí_li};
    Konec;
  00Eh: { SO }
    Začátek
      GZ.Široké_Do_Konce_Řádku := pravda;
    Konec;
  00Fh: { SI }
    Začátek
      GZ.Zhuštěný := pravda;
    Konec;
  012h: { DC2 }
    Začátek          GZ.Zhuštěný := nepravda;
    Konec;
  014h: { DC4 }
    Začátek
      GZ.Široké_Do_Konce_Řádku := nepravda;
    Konec;
  018h: { CAN }
    Začátek
      RVP.Smaž_Všechno;
    Konec;
  01Bh: { ESC }
    Začátek
      Zpracuj_ESC;
    Konec;
  v_ostatních_případech { neznámý řídicí znak }
    Začátek
      Platí_li NUL_CAN_Řídicí tak
        RVP.Tiskni_Kus_Řádku;
        RU.Přidej_Do_VP (OU.Nezdar);
      jinak
        GZ.Zpracuj_Znak (Přečtený_Znak);
      Konec {Platí_li};
    Konec;
  Konec {Vyber};
jinak_platí_li (Přečtený_Znak = 07Fh)
    nebo (Řídicí_Taky_Přes_80h
    a_zároveň (Přečtený_Znak = 0FFh)) tak
  RVP.Smaž_Znak;
jinak { tištitelný znak }
  GZ.Zpracuj_Znak (Přečtený_Znak);
```

```
    Konec {Platí_li};
  až_bude_platit nepravda;
Konec {Zpracuj_Znak};

{*************************************************************}
Konec { ZZ - zpracování znaků }.
```

Podívejme se na uvedený program. Pozornému čtenáři jistě neušlo, že se v modulu ZZ objevila další procedura, Zpracuj_ESC. Nedůvěřivý čtenář snad podezírá autora, že si svévolně vymýšlí nové a nové procedury a že dělá vědu z věcí, bez kterých bychom se klidně mohli obejít. Vždyť kdybychom každý příkaz volání procedury nahradili skupinou příkazů obsažených ve volané proceduře, byl by výsledek stejný - a hlavně bez procedur!

Jenže kdo se v tom pak má vyznat! Autor znovu podotýká, že procedurami se dosahuje především srozumitelnosti programů. Autor si vymyslel novou proceduru vždycky, když se při programování začal zaplétat do podrobností a přestávalo mu být jasné, k čemu slouží to, co právě programuje. Předčasným vypracováváním podrobností se prostě ztrácí přehled, vytrácí se smysl, mizí koncepce. Často se pak několikrát opisuje jeden a tentýž úsek na několika místech v programu. To je ovšem příznak hrubého a pokročilého zanedbání rozboru úlohy. Je nad slunce jasné, že takový úsek programu by měl být procedurou a neměl by se v programu opakovat, nýbrž by měl být opakovaně volán.

Správným rozkladem programu na procedury vzniká vrstevnatá struktura. Je to podobná struktura jako vrstevnatá struktura modulů při rozkladu úlohy. Rozdíl je v tom, že moduly jsou celky tvořené procedurami i daty a navzájem jsou spojeny dovozem a vývozem objektů, kdežto procedury jsou jen pracovní postupy (popisy činností) a nemají možnost dovozu a vývozu objektů (při používání objektů se řídí přísnými pravidly soukromého vlastnictví).

Musíme si ještě ujasnit, jak složité procedury budeme psát a jak velké množství procedur připustíme. Řekli jsme si, že složité činnosti budeme vysvětlovat pomocí jednodušších příkazů. Čím jednodušší příkazy při vysvětlování použijeme, tím bude procedura složitější.

Jak tedy odpovíme na otázku počtu a složitosti procedur? Napoleon to řešil číslem sedm (v každé vrstvě). Pan Ing. Ivan Slota z Košic doporučuje psát jen tak dlouhé procedury, které nám nebude líto vyhodit, až budeme předělávat program. Musíme si však uvědomit, že příliš krátké procedury vedou ke vzniku příliš mnoha vrstev - vznikne nepřehledný systém složený z přehledných prvků, který je konec konců stejně nepoužitelný jako přehledný systém složený z nepřehledných prvků.

Nyní zbývá ještě vysvětlit některé nové pojmy a příkazy, které se v modulu ZZ vyskytly a nad kterými si pozorný čtenář možná marně láme hlavu.

V proceduře Čti_Znak se vyskytuje příkaz:

```
Přečtený_Znak := RZ.Vyjmi_Z_VP;
```

Tento příkaz poznáme podle značky ":=". Nalevo od ní musí stát označení proměnné a napravo hodnota (výraz). Příkaz způsobí, že hodnota bude uložena do proměnné. V našem případě bude zavolána funkce Vyjmi_Z_VP z modulu RZ, která vrátí hodnotu znaku přečteného z nadřízeného počítače. Tato hodnota pak bude uložena do proměnné Přečtený_Znak.

Tomu, co jsme si právě vysvětlili, se poněkud vymyká příkaz "Řídicí := <výraz>;" uvedený ve funkci Řídicí. Bystrému čtenáři snad neušlo, že na levé straně od znaménka ":=" nestojí jméno proměnné, nýbrž jméno funkce (a to dokonce té funkce, ve které je příkaz použit). Zmíněná podivuhodnost souvisí s tím, že vykonáním funkce vznikne hodnota. Dosud však zůstalo čtenáři utajeno, jak se uvnitř v deklaraci funkce napíše, že zrovna ta určitá hodnota má být výsledkem funkce. Představme si, že jménem funkce nebudeme označovat pouze skupinu příkazů (jak je tomu u obyčejné procedury), nýbrž navíc budeme jménem funkce označovat i proměnnou, do které funkce uloží výslednou hodnotu. Zavoláním funkce se jednak vykonají její příkazy, jednak se vyjme hodnota z oné proměnné. Uvnitř funkce můžeme do zmíněné proměnné hodnoty ukládat s tím, že hodnota uložená jako poslední bude výsledkem funkce.

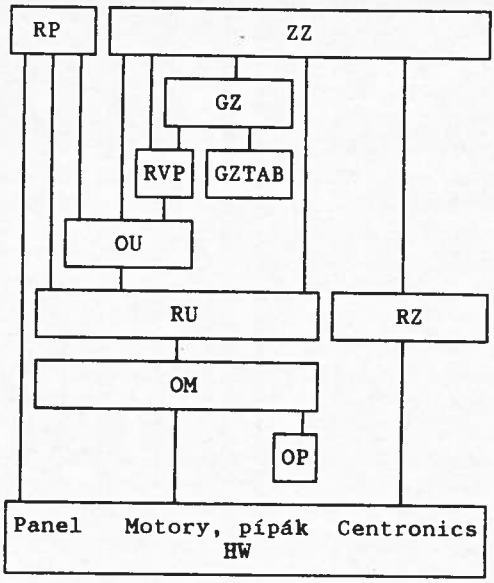
V zadání úlohy jsme uvedli, že některé počítače posílají do tiskárny jen sedmibitové údaje (místo obvyklých osmibitových bajtů). Rozhraní znaků však předává do modulu ZZ vždy celé bajty s tím, že nejvýznamnější bit v bajtu může být neplatný. Hodnota nejvýznamnějšího bitu se pak nastaví podle řídicích posloupností ESC '>' nebo ESC '='. Abychom však mohli pracovat s jednotlivým bitem, nemůžeme použít typ bajt (tj. číslo od 0 do 255), nýbrž množinu - osmici bitů. Operací + můžeme do jedné množiny přidat prvky obsažené ve druhé množině, operací * můžeme v jedné množině ponechat jen ty prvky, které jsou současně obsaženy i v druhé množině, a ostatní prvky tak odstranit. K tomu však potřebujeme nějak vyjádřit, že proměnná Přečtený_Znak, která je normálně typu bajt, má být použita jako množina typu Osmice_Signálů. A toho dosáhneme poněkud nezvyklým použitím slova **Typ**, které v tomto případě neoznačuje deklaraci typu, nýbrž dočasnou změnu typu proměnné:

```
Přečtený_Znak Typ HW.Osmice_Signálů :=
  (Přečtený_Znak Typ HW.Osmice_Signálů +
  Nastav_Bit_8 Typ HW.Osmice_Signálů) *
  Zruš_Bit_8 Typ HW.Osmice_Signálů;
```

Typ Osmice_Signálů jsme dovezli do modulu ZZ z modulu HW. Modul HW popisuje technické vybavení podle zadání (kapitola 3.4.5.). Deklaraci modulu HW najde zvídavý čtenář o několik stránek dále.

Kromě nového modulu HW jsou v programu používány objekty dalších dvou nových modulů: OU a RU. Zkratka OU znamená "ovladač úkonů" a RU znamená "rozhraní úkonů". Co je to úkon a proč autor vůbec zavádí tento nový pojem? Bezprostřední ovládání mechanických částí tiskárny je pro moduly ZZ, RVP a RP příliš složité. Znamenalo by to nejen bezprostřední práci s porty, ke kterým jsou připojeny motory, hlava a pípák, ale i časování (tj. řízení rychlosti), rozjezdy a brzdění motorů, práci s výškou a délkou jednotlivých tónů pro pípák apod. Tyto činnosti proto dáme na starost zvláštnímu modulu "ovladač mechaniky" (OM), který bude vykonávat jednotlivé jednoduché úkony podle pokynů z ostatních modulů. Jednotlivé úkony budeme posílat ovladači mechaniky prostřednictvím modulu RU. Některé činnosti (např. řádkování a stránkování) se však provádějí složitějším způsobem než pouhým vysláním pokynu k vykonání úkonu. A právě tyto složitější záležitosti bude mít na starost modul OU.

Tím dostaneme úplnou soustavu modulů, která nám umožní splnit zadání úlohy poměrně snadným a přehledným způsobem. Tato soustava je na obrázku 4.2.



Obr. 4.2

Uvnitř příkazu na opakované zpracování znaku v proceduře Zpracuj_Znak se setkáváme s dosud neznámým příkazem

```
Vyber <výraz> ze
<konstanta>:
  Začátek
    <příkaz>
      ...
    <příkaz>
  Konec;
<jiná konstanta>:
  Začátek
    <příkaz>
      ...
    <příkaz>
  Konec;
    ...
<jiná konstanta>:
  Začátek
    <příkaz>
      ...
    <příkaz>
  Konec;
v_ostatních_případech
  Začátek
    <příkaz>
      ...
    <příkaz>
  Konec;
Konec {Vyber};
```

Tento příkaz se skládá z několika skupin příkazů. Každá skupina je označena jinou konstantou než ostatní skupiny. Konstanty jsou stejného typu jako výraz za slovem **Vyber**. Nejprve se vyhodnotí výraz za slovem **Vyber**. Pak se provede skupina příkazů označená tou konstantou, která má stejnou hodnotu jako vyhodnocený výraz. Budeme-li např. zpracovávat znak FF s kódem 00Ch, bude hodnota výrazu 00Ch a provede se skupina příkazů označená konstantou 00Ch (tj. zavolá se procedura Stránkuj). Může však dojít i k tomu, že vyjde hodnota výrazu, která neodpovídá žádné z konstant. V takovém případě se vykoná skupina příkazů označená **v_ostatních_případech**. Pokud takto označená skupina příkazů vůbec není uvedena, neudělá se jednoduše vůbec nic.

A nakonec je na místě, abychom stručně prodiskutovali, proč procedura Čti_Znak používá proměnnou Přečtený_Znak. V myslích některých čtenářů možná hlodá pochybnost, zda procedura Čti_Znak má skutečně přístup k proměnné Přečtený_Znak. A pisatel praví: ano, má. Neboť všechny později deklarované soukromé objekty procedury Zpracuj_Znak mohou používat všechny dříve deklarované soukromé objekty procedury Zpracuj_Znak. Proceduru Čti_Znak jsme

deklarovali později než proměnnou Přečtený_Znak, a proto Čti_Znak smí používat proměnnou Přečtený_Znak.

Vtipného čtenáře možná napadlo, že procedura Čti_Znak by mohla být deklarována

```
Funkce Čti_Znak: bajt;
```

a použili bychom ji v příkazu

```
Přečtený_Znak := Čti_Znak;
```

Tím by se odstranil nebezpečný postranní účinek procedury Čti_Znak. Ta totiž v našem programu mění obsah proměnné Přečtený_Znak, aniž by to bylo nějak zřetelně vyjádřeno ve volajícím programu. Původní procedura Čti_Znak obsahuje pevnou vazbu na proměnnou Přečtený_Znak a tuto vazbu bychom zmíněnou úpravou odstranili (hodnotu přečteného znaku bychom pak mohli ukládat do libovolné proměnné nebo ji použít jako součást výrazu a neukládat ji vůbec, zatímco proměnnou Přečtený_Znak bychom mohli použít k uchovávání jakýchkoli hodnot - tedy nejen k uchovávání hodnoty přečteného znaku).

Ale je právě tohle naším cílem? Nikoli. My potřebujeme na určitých místech v programu přečíst další znak (aniž bychom zároveň použili jeho hodnotu) a na jiných místech v programu potřebujeme hodnotu přečteného znaku několikrát použít. A právě proto zvolíme původní způsob čtení znaků. Vyjádříme tím svůj záměr jasně a přiměřeně za cenu určitého nebezpečí. Abychom se vyvarovali chyb, zavedeme několik bezpečnostních pravidel, která budeme přísně dodržovat:

- proměnná Přečtený_Znak nesmí obsahovat jinou hodnotu než hodnotu posledně přečteného znaku,
- obsah proměnné Přečtený_Znak smí být měněn jedině procedurou Čti_Znak (a žádným jiným způsobem).

Vidíme, že proměnná Přečtený_Znak a procedura Čti_Znak tvoří funkční celek, který bychom měli vyjádřit v podobě samostatného modulu. To však neuděláme z čistě praktických důvodů: takový přístup by vedl ke vzniku příliš velkého počtu příliš malých modulů.

Autor doporučuje laskavé pozornosti čtenáře, že proměnná Přečtený_Znak se jmenuje právě Přečtený_Znak (a ne třeba PQ17) a že smí obsahovat jedině posledně přečtený znak (a ne třeba porodní hmotnost prababičky Františka Palackého). Pokud chceme omezit počet chyb, kterých se při programování dopustíme, na únosnou míru, je bezpodmínečně nutné přesně vymezit význam každé proměnné a toto vymezení pak přísně dodržovat. Tomu napomáhá jednak výstižné jméno proměnné, jednak volba vhodného typu (jinak bychom mohli označovat proměnné jejich adresami v paměti a používat univerzální osmibitový typ).

Někdy se však stane, že není možné dodržet význam proměnné. Kdybychom například chtěli počítat, kolik znaků tiskárna vytiskla, museli bychom zároveň s vytištěním každého znaku zvětšit počitadlo vytištěných znaků o jedničku. Zároveň to ale nejde, musí se to dělat postupně: buďto napřed vytisknout, tím bude mít počitadlo nesprávnou hodnotu, a pak teprve zvětšit počitadlo, anebo napřed zvětšit počitadlo (tím bude mít nesprávnou hodnotu), a pak teprve vytisknout znak.

Takovým nebezpečným místům v programu se nevyhneme. Je však důležité, abychom dokázali nebezpečí omezit na rozumnou míru. Proto se snažíme, aby nebezpečné úseky programu byly co nejkratší, abychom proměnným co nejdříve vrátili jejich správný význam. A za druhé dbejme vždy na to, aby se nebezpečná místa neprolínala navzájem; abychom nezačínali nebezpečný úsek programu dřív, než dokončíme předchozí nebezpečný úsek. Je vhodné vytvářet z nebezpečných míst samostatné procedury nebo je alespoň zřetelně označit komentářem.

Nejhorší případ nastane, když nebezpečné místo souvisí se sdílením nějakého prostředku několika procesy. Lehce nahlédneme, že sdíleným prostředkem nemusí být jen výše zmíněný procesor. Procesy mohou sdílet proměnné (když si potřebují navzájem předávat nějaké hodnoty) nebo mohou sdílet třeba procedury (např. procedura OU.Stránkuj je sdílená procesy popsanými v modulech ZZ a RP).Mimořádné nebezpečí hrozí tehdy, když je nebezpečné místo v proceduře, kterou sdílí několik procesů, nebo když jsou v několika procesech nebezpečná místa, kde se porušuje význam téže sdílené proměnné. Procesy si pak porušují význam sdílené proměnné navzájem jeden druhému. V takovém případě dochází ke zvlášť záludným a těžko odstranitelným chybám. Program s chybou tohoto druhu může dávat pokaždé jiné výsledky, někdy se zasekne uprostřed a vůbec neskončí a jindy zase pracuje úplně správně (a zkuste hledat chybu v programu, který náhodou zrovna pracuje správně). Těmto chybám můžeme předcházet dodržováním určitých pravidel, která stanovíme později, až se budeme zabývat sdílením prostředků několika procesy.

Nyní se vraťme k modulu RP:

```
Modul RP;
{ RP - Rozhraní vrchního panelu }
{ Ivan Ryant, 25.06.88/16:05 }

Vývoz
  Kontroluj_Papír, Nastav, Ošetři;
Dovoz
  HW, RU, OU;

Konstanta
  Tl_Ruční      = S3; { přepíná provoz ruční / spřažený }
  Tl_Posun      = S4; { ruční posun papíru }
```

```
  Světlo_Ruční = S7; { světýlko "RUČNÍ" }
  Došel_Papír  = S2; { snímač konce papíru }

Proměnná
  Kontroluj_Papír: logický;

{************************************************************}

Funkce Nastav: logický;
{ počáteční nastavení - vrací, zda se má tisknout pečlivě }
Proměnná
  i: bajt;
Začátek
  Kontroluj_Papír := pravda;
  Platí_li Tl_Posun je_prvkem HW.Stav tak
    Nastav := pravda;
    Přidej_Do_VP (OU.Zvonek);
  jinak
    Nastav := nepravda;
  Konec {Platí_li};
Konec {Nastav};

{************************************************************}

Procedura Ošetři;
{ ošetřuje vrchní panel, předpokládá pravidelné volání }
{ od normy EPSON se liší tím, že nevysílá signál ERR   }
Proměnná
  i: celý;
{============================================================}

Funkce Chybí_Papír: logický;
Začátek
  Chybí_Papír := RP.Kontroluj_Papír a_zároveň
    (Došel_Papír je_prvkem HW.Stav);
Konec {Chybí_Papír};

{************************************************************}

Začátek
  Platí_li (Tl_Ruční je_prvkem HW.Stav) nebo (Chybí_Papír) tak
    HW.Řízení_8255 := 2 * ord (Světlo_Ruční) + 1; {rozsviť}
    Odtud_opakuj
    až_bude_platit neplatí (Tl_Ruční je_prvkem HW.Stav);
    Platí_li (Chybí_Papír) tak
      Přidej_Do_VP (OU.Došel_Papír);
    Konec {Platí_li};
    Odtud_opakuj
      Platí_li Tl_Posun je_prvkem HW.Stav tak
```

```
        i := 15000; { krát 70 us }
        Odtud_opakuj
          i := i - 1;
        až_bude_platit (i = 0) nebo
          neplatí (Tl_Posun je_prvkem HW.Stav);
        { buď vypršel čas nebo bylo puštěno tlačítko }
        Platí_li Tl_Posun je_prvkem HW.Stav tak
          OU.Stránkuj;
        jinak
          OU.Řádkuj;
        Konec {Platí_li};
      Konec {Platí_li};
    až_bude_platit (Tl_Ruční je_prvkem HW.Stav) a_zároveň
      neplatí (Chybí_Papír);
    Odtud_opakuj
    až_bude_platit neplatí (Tl_Ruční je_prvkem HW.Stav);
    HW.Řízení_8255 := 2 * ord (Světlo_Ruční) + 0; { zhasni }
  Konec {Platí_li};
Konec {Ošetři};

{*************************************************************}

Procedura Ošetřuj_Panel;
Začátek
  Odtud_opakuj
    Ošetři;
  až_bude_platit nepravda;
Konec {Ošetřuj_Panel};

{*************************************************************}

Konec { RP - rozhraní vrchního panelu }.
```

Uvedený zápis modulu RP neobsahuje nic nového ve srovnání s modulem ZZ. Autor doporučuje pozornosti čtenáře jen jedinou věc: mechanická část tiskárny je (podobně jako v modulu ZZ) ovládána pomocí ovladače úkonů a rozhraní úkonů. Kromě toho však modul RP také pracuje přímo s technickým vybavením tiskárny. Nejedná se však o libovolné části technického vybavení, nýbrž o technické vybavení vrchního panelu tiskárny. To ovšem můžeme považovat za projev zdravého rozumu, že program ošetřující vrchní panel pracuje přímo s vrchním panelem.

Nyní ještě zbývá uvést slibovaný modul HW:

```
Definiční modul HW;
{ Popis rozhraní technických prostředků }
{ Ivan Ryant, 25.06.88/17:40 }

{ Tento modul slouží modulům ve vyšších vrstvách, které  }
```

```
{ přímo používají technické vybavení nebo typy dat defi- }
{ nované na technickém vybavení; vše, co je zde popsáno, }
{ se automaticky vyváží                                  }

Port
  Data                 = 00h,
  Přepínače_Předvolby = 30h,

 { Časovač 8253 }
  Hodiny       = 10h,
  Výška_Tónu   = 11h,
  Délka_Tónu   = 12h,
  Řízení_8253  = 13h,

 { Paralelní vstup/výstup 8255 }
  Stav         = 20h,
  Spodní_Jehly = 21h,
  Řízení       = 22h,
  Řízení_8255  = 23h;

Konstanta
 { adresy dalších portů }
    { k vykonání uvedených činností je třeba přečíst port }
    { nebo do něj zapsat, na datech nezáleží }
  Vpravo = 40h;  { krok tiskací hlavy doprava }
  Vlevo  = 41h;  { krok tiskací hlavy doleva  }
  Dolů   = 42h;  { krok válce vpřed, aby se tisklo níž }
  Nahoru = 43h;  { krok válce zpět, aby se tisklo výš }
  Bum    = 44h;  { úder jehel 0.7 až 1 ms }

Typ

  Signál = (S0, S1, S2, S3, S4, S5, S6, S7);
  Osmice_Signálů = množina_nad Signál;
  Jehla = (J1, J2, J3, J4, J5, J6, J7, J8); { spodní jehly }
  Osmice_Jehel = množina_nad Jehla;
Typ
  Přepínač  =
    (IBM, KOI, Papír_12, Přeskok_Perforace);
  Předvolba = množina_nad Přepínač;

Proměnná
  Data:      bajt; { vstupní data z rozhraní Centronics }
  Přepínače_Předvolby: Předvolba; { 4 přepínače }

 { Časovač 8253 }
  Hodiny,     { každou 1 ms způsobí NMI - tiknou }
  Výška_Tónu, { tvoří symetrický signál pro pípák }
  Délka_Tónu, { vpustí signál do pípáku po zadanou dobu }
```

```
  Řízení_8253: bajt; { řízení obvodu 8253 }

 { Paralelní vstup/výstup 8255 }
  Stav:         Osmice_Signálů;
  Spodní_Jehly: Osmice_Jehel;
  Řízení:       Osmice_Signálů;
  Řízení_8255: bajt; { řízení obvodu 8255 }

{************************************************************}
```

Konec { HW - rozhraní technických prostředků }

V modulu HW zasluhuje vysvětlení snad jen práce s porty. V kapitole 2 jsme si řekli, že port je vlastně jen zvláštním druhem proměnné. Do některých portů se dá jen zapisovat a z jiných se zase dá jen číst. A obecně platí, že z portu nemusíme přečíst to, co jsme do něj předtím byli zapsali.

Ale nic nám nebrání, abychom s porty zacházeli jako s proměnnými: můžeme je deklarovat i používat stejnými příkazy jako všechny ostatní proměnné.

Je tu však jeden rozdíl: zatímco u proměnných nám obvykle nezáleží, na kterých adresách se budou nacházet (a obvykle se tím vůbec ani nechceme zatěžovat), u portů máme předepsáno v notách, jaké mají mít adresy. A právě to zapíšeme v programu popisem **Port**. Za slovem **Port** pak uvedeme seznam všech jmen portů s příslušnými adresami.

Některé porty však nebudeme používat jako proměnné. Jsou to porty, které umožňují vykonávat kroky motorů vpřed nebo vzad. Adresy těchto portů jsou nadeklarovány jako konstanty, protože je budeme pouze předávat jako parametry do zvláštních procedur napsaných v jazyce procesoru. Tyto procedury pak jednoduše a rychle vykonají požadovaný krok motoru. Samozřejmě bychom totéž mohli udělat i zápisem do neexistující proměnné umístěné na adresu příslušného portu, ale nemáme důvod vyjadřovat nesrozumitelným, nepřirozeným a nepřiměřeným způsobem, pomocí programátorské finty, něco, co lze stručně a jasně zařídit přiměřenými prostředky na nižší úrovni abstrakce, nikoli na úrovni rozboru úlohy, nýbrž na úrovni zápisu programu, na úrovni jazyka procesoru.

Shrnutí:

Zavedli jsme pojmy proces a sdílený prostředek. Stanovili jsme úplnou vrstvenou strukturu modulů pro náš program. Zmínili jsme se o nebezpečných místech v programu.

4.4. Několik slov o záchodové míse

V minulé kapitole jsme si řekli, že procesy popsané v modulech ZZ a RP sdílejí společný prostředek - procesor. Vidíme však, že to není jediný prostředek, který zmíněné procesy sdílejí. Pohledem do našich programů snadno zjistíme, že jak procedura Ošetři, tak i procedura Zpracuj_Znak používá příkazy Řádkuj a Stránkuj.

Kdybychom se podívali dovnitř do procedur Řádkuj a Stránkuj, našli bychom v nich nebezpečná místa, o kterých jsme se zmínili v předchozí kapitole. Jenže právě tohle my neuděláme. Vždyť jsme teprve na počátku rozboru úlohy, tak by bylo chybou příliš zabíhat do podrobností. Můžeme však stanovit obecná pravidla pro sdílení prostředků. Tato pravidla jsou sice jednoduchá, ale poměrně přísná, protože musí platit vždy a všude. Většinou se jim můžeme snadno podřídit. V některých jednotlivých případech však lze stanovit pravidla mírnější - ale je potom nutné znát vnitřní stavbu sdíleného prostředku.

Co se stane, když bude jak procedura Ošetři, tak Zpracuj_Znak najednou řádkovat nebo stránkovat? Vůbec nelze určit, kolikrát se odřádkuje nebo odstránkuje, a navíc to v každém jednotlivém případě může dopadnout jinak.

Je to podobné, jako když se dva lidé pokoušejí použít ve stejném okamžiku společnou záchodovou mísu. Oba najednou se na mísu nevtěsnají a ani způsob "každý chvilku tahá pilku" nevede k uspokojivým výsledkům. Záchodová mísa totiž patří mezi takové sdílené prostředky, které vyžadují **výlučný přístup**. Každý uživatel musí počkat, až ostatní uživatelé prostředek **uvolní**, teprve pak jej může sám **obsadit**, použít (s vyloučením ostatních uživatelů) a po dokončení operace (ovšem v konečné době) uvolní prostředek pro další uživatele.

Pro úplnost poznamenejme, že latrína, o které se zmiňuje Hašek v Osudech dobrého vojáka Švejka, byla také sdíleným prostředkem, umožňovala však mnohonásobný přístup. Viz

J.Hašek: Osudy dobrého vojáka Švejka za světové války,
díl III. Slavný výprask, strana 114,
Československý spisovatel, Praha, 1980.

Ale i v tomto případě mohlo dojít k zaplnění sedadla latríny a k vyloučení dalších uživatelů (tj. k pozdržení jejich činnosti).

Konec konců i procesor je prostředek, který vyžaduje výlučný přístup. Tady však dochází ke zjednodušení: kdo obdrží procesor, získává tím i právo výlučného přístupu k němu. A kdo procesor

nemá, nemá ani možnost přístupu k procesoru. Je to jako ve fotbale: kdo nemá míč, nemůže dát gól.

Místo v programu, kde se přistupuje ke sdílenému prostředku, nazveme **kritické místo**. Kritické místo je třeba ošetřit zajištěním výlučného přístupu ke sdílenému prostředku. Takto ošetřené kritické místo budeme nazývat **kritická sekce**.

Z uvedené teorie vyvodíme důsledky pro řádkování a stránkování. Procedury Řádkuj a Stránkuj můžeme považovat za sdílené prostředky, které jsou sdíleny procedurami Ošetřuj_Panel a Zpracuj_Znak. Procedury Ošetřuj_Panel a Zpracuj_Znak musí obsahovat kritické sekce, kterými si příslušné procesy zajistí výlučný přístup ke sdíleným procedurám Řádkuj a Stránkuj. Kritické sekce můžeme naprogramovat různě složitými způsoby, které se navzájem liší svou použitelností (později se setkáme ještě s dalšími, složitějšími způsoby sdílení prostředků několika procesy). Nyní se nám naskýtá možnost velmi jednoduchého řešení.

Kdo má procesor, má i výlučný přístup ke všem prostředkům. Podstatou kritické sekce tedy je vhodný způsob předávání procesoru mezi procesy, Jeden mechanismus předávání procesoru již známe: je to mechanismus volání procedury. Jeden z našich dvou procesů bude vlastníkem procesoru a bude procesor půjčovat druhému procesu. Jak toho dosáhneme?

Procedura Ošetřuj_Panel opakovaně volá proceduru Ošetři. Můžeme si však docela dobře představit, že procedura Ošetři bude opakovaně volána z procedury Zpracuj_Znak z modulu ZZ. Proceduru Ošetřuj_Panel nyní můžeme úplně vynechat.

```
{ Cyklus na zpracování znaku }
 Odtud_opakuj
   Platí_li RZ.Je_Znak_Ve_VP tak
     RP.Ošetři;
   jinak
     Platí_li Bezprostřední_Tisk tak
       RVP.Tiskni_Kus_Řádku;
       OU.Posuň_Papír_Na (OU.Poloha_Papíru + 216);
       OU.Vykonej_Posuny (0);
     Konec {Platí_li};
     Odtud_opakuj
       RP.Ošetři;
     až_bude_platit RZ.Je_Znak_Ve_VP;
     Platí_li Bezprostřední_Tisk tak
       OU.Posuň_Papír_Na (OU.Poloha_Papíru - 216);
     Konec {Platí_li};
   Konec {Platí_li};

  { Zpracuj znak z rozhraní znaků }
   Čti_Znak;
   ...
 až_bude_platit nepravda;
```

Shrnutí:

Vysvětlili jsme pojmy výlučný přístup, kritické místo a kritická sekce. Čtenář by měl chápat rozdíl mezi nebezpečným místem, kritickým místem a kritickou sekcí.

4.5. Složité typy dat

Při psaní modulů ZZ a RP jsme použili několika modulů, které budeme muset teprve napsat. Jedná se především o moduly RVP, GZ, RZ a RU. Tyto moduly mají jedno společné: jádrem každého z nich bude složitě uspořádaná (neboli strukturovaná) proměnná. Přesněji řečeno, proměnná složitě strukturovaného typu. Potřebné operace s touto proměnnou bude provádět několik procedur, které budou součástí příslušného modulu.

Bystrého čtenáře snad překvapí nápadná (a nikoli náhodná) podobnost s pojmem "typ": modul jednak určuje, jaké (složitě strukturované) hodnoty můžeme do zmíněné složité proměnné ukládat, jednak určuje, jaké operace můžeme s těmito hodnotami provádět. Rozdíl je v tom, že nechceme samostatně deklarovat typ celé skupiny proměnných (které bychom pak časem postupně deklarovali), nýbrž chceme deklarovat přímo jednu určitou proměnnou požadovaného složitého typu. Samozřejmě je však také možné deklarovat v modulu samotný typ (a nikoli proměnnou) a vyvážet jej do ostatních modulů, kde se teprve deklarují proměnné onoho typu - ale to my nepotřebujeme.

V podobě strukturovaných proměnných vytvoříme např. paměť na řádek znaků a generátory znaků. Musíme naprogramovat jednak jejich strukturu, jednak operace s jejich hodnotami. Jednotlivé operace budeme programovat jako procedury. Například udělame zvláštní proceduru pro přidání znaku do řádkové paměti, jinou procedurou budeme znaky z řádkové paměti vyjímat a předávat k tisku.

A jak vytvoříme strukturu proměnné (tj. strukturu dat, která mohou být v proměnné uložena)? Data, která mají složitou strukturu, musíme rozložit na jednodušší prvky. Jednodušší prvky mohou být buď zase strukturované (a pak je musíme zase rozložit), anebo mohou být jednoduchého typu (např. bajt, znak, nějaký množinový typ apod.). Skládání prvků do struktur lze provádět dvěma způsoby: prvky stejného typu vytvářejí tzv.

pole (viz tab 4.1),

zatímco prvky různých typů vytvářejí tzv.

záznam (viz tab. 4.1).

V tabulce 4.1 vidíme, že prvkům pole (které jsou stejného typu) je přidělována stejně veliká paměť, kdežto různým prvkům záznamu může být přidělena různě veliká paměť (v závislosti na typech prvků). V tabulce 4.1 je 1 bajt znázorněn jednou pomlčkou. Prvky pole se rozlišují pomocí tzv.

indexů,

tj. souvislé posloupnosti hodnot nějakého typu (jako indexy nám mohou posloužit například pořadová čísla prvků v poli), kdežto prvky záznamu se rozlišují pomocí jmen (každý prvek v záznamu má své jméno, které je soukromým jménem v rámci příslušného záznamového typu).

položka pole nebo záznamu	velikost přidělené paměti
GZ.Gen [GZ.ROM] [0].Popis_Tvaru	-
GZ.Gen [GZ.ROM] [0].Tvar_Znaku [0]	-
GZ.Gen [GZ.ROM] [0].Tvar_Znaku [1]	-
GZ.Gen [GZ.ROM] [0].Tvar_Znaku [2]	-
GZ.Gen [GZ.ROM] [0].Tvar_Znaku [3]	-
GZ.Gen [GZ.ROM] [0].Tvar_Znaku [4]	-
GZ.Gen [GZ.ROM] [1].Popis_Tvaru	-
GZ.Gen [GZ.ROM] [1].Tvar_Znaku	-----
GZ.Gen [GZ.ROM] [2].Popis_Tvaru	-
GZ.Gen [GZ.ROM] [2].Tvar_Znaku	-----
GZ.Gen [GZ.ROM] [3]	------
GZ.Gen [GZ.ROM] [4]	------
...	
GZ.Gen [GZ.ROM] [255]	------
GZ.Gen [GZ.RWM] [0]	------
...	
GZ.Gen [GZ.RWM] [255]	------

Tab. 4.1

Prohlédněme si modul GZ. Znaky se generují v proceduře GZ.Zpracuj_Znak. Podle kódu požadovaného znaku se pomocí generátoru znaků vytvářejí údaje pro tisk jednotlivých sloupců a ty se ukládají do řádkové paměti. Generátory znaků ROM-generátor a RWM-generátor jsou uspořádány v proměnné Gen v podobě pole. Pole Gen se skládá ze dvou položek typu GZ.Generátor: z ROM a RWM generátoru. Tyto dvě položky lze navzájem rozlišit pomocí indexů, které mohou nabývat hodnot typu GZ.Druh, tj. jedné ze dvou hodnot - buď GZ.ROM, anebo GZ.RWM. V deklaraci pole uvádíme za slovem **pole** typ indexů v hranatých závorkách, např. **pole** [GZ.Druh] **ze** GZ.Generator. Když potom chceme v nějakém příkazu použít prvek pole, napíšeme jméno proměnné typu pole a potřebnou hodnotu indexu v hranatých závorkách, např. Gen [GZ.ROM].

Typ GZ.Generátor je opět pole, které se skládá z 256 položek typu GZ.Položka. Položky jsou rozlišeny pomocí indexů, které nabývají hodnot všech možných kódů znaků, takže každý kód znaku má v generátoru svou položku. Znaku 'A' např. odpovídá položka Gen [GZ.ROM] [41h], což můžeme zapsat také zjednodušeným způsobem Gen [GZ.ROM, 41h].

Typ GZ.Položka je opět strukturovaný typ. Není to však pole, nýbrž záznam. Tento záznam se skládá ze dvou prvků: první z nich se jmenuje Popis_Tvaru, druhý se jmenuje Tvar_Znaku. Popis_Tvaru obsahuje informace o tom, zda se má tisknout spodními nebo

vrchními osmi jehlami, a číslo prvního a posledního sloupce, který se má tisknout. Tvar_Znaku obsahuje množiny popisující, které černé tečky se mají udělat. Tvar_Znaku je ještě dále strukturován - je to pole pěti osmiprvkových množin. Položky záznamu používáme v příkazech tak, že nejprve napíšeme jméno záznamu, pak tečku a jméno prvku, např.: Gen [GZ.ROM] [68].Popis_Tvaru.

Zůstává ještě nezodpovězenou otázkou, jak se do proměnné Gen dostane správný obsah, tj. generátory znaků. Autor to vyřešil tak, že napsal pomocný modul GZTAB, který obsahuje vzorovou hodnotu pro všechny generátory znaků. Modul GZTAB dále obsahuje pomocné tabulky GZTAB.KOI_8_čs, GZTAB.Přeskupení a GZTAB.Národní_Sada.

Vzorový generátor a pomocné tabulky jsou místa v paměti, na která je již při programování uložena pevná hodnota - a ta se při běhu programu nesmí měnit. Tato místa v paměti můžeme považovat za zvláštní proměnné, do kterých program nesmí ukládat žádné hodnoty a jejichž počáteční hodnota je předem pevně nastavena. Modul GZTAB je napsán v jazyce procesoru 8080, na kterém autor celý program vyzkoušel. Zvídavý čtenář jej najde v kapitole 5 a jistě snadno nahlédne, že kromě několika komentářů tam jsou postupně uvedeny hodnoty všech bajtů, které je potřeba nastavit.

Podle pomocných tabulek lze přeskupit obsah vzorového generátoru GZ.IBM do podoby generátoru kterékoli národní sady znaků. Generátor zvolené sady znaků se přesune do proměnné Gen [GZ.ROM] ze vzorového generátoru GZ.IBM podle "návodu" uloženého v pomocných tabulkách.

Dříve než si čtenář samostatně prohlédne návrh modulu GZ v kapitole 5, měl by znát ještě jednu věc: Okénko pro tisk znaku má celkem 12 sloupců po 24 linkách, ale položka generátoru obsahuje data jenom pro pět osmic jehel, tj. pro 5 sloupců po osmi linkách. Musíme tedy zodpovědět zásadní otázku - jak se z pěti sloupců po osmi linkách udělá dvanáct sloupců po 24 linkách?

Především si musíme uvědomit, že ne vždy tiskneme do všech políček v okénku, ba naopak - obvykle necháváme určité oblasti v okénku bílé. Víme, že při ledabylém tisku se píše jen ob linku a ob sloupec. Pouze znaky tučné nebo pečlivé se tisknou do všech sloupců a linek v okénku.

V kapitole 3 jsme se však dohodli, že nedokážeme nastavit tiskací hlavu přesně do všech sloupců a linek okénka, že můžeme počítat s přesností asi jen poloviční (tedy ob linku a ob sloupec). To znamená, že tvar znaku musí být vyjádřen 6 sloupci po 12 linkách, protože jemnější podrobnosti by se nepodařilo zachytit na papír. Proto při tučném tisku budeme do těch linek a sloupců, které v generátoru nejsou, opakovat obsah sousedních linek a sloupců (které v generátoru jsou). A při pečlivém tisku doplníme do volných políček černé tečky tak, abychom spojili do souvislých černých skvrn všechny sousedící tečky v generátoru.

To znamená: sousedí-li v generátoru dvě tečky (vodorovně, svisle nebo úhlopříčně), bude mezi ně vytištěna další černá tečka (viz obr. 4.3).

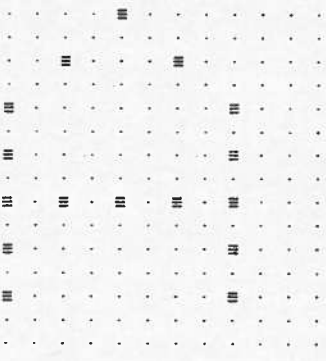
Takže je zřejmé, že vystačíme s generátorem pro políčko se 6 sloupci po 12 linkách. Tyto nároky však můžeme ještě více omezit. Můžeme rozlišit dva druhy znaků. Znaky, které slouží k vytváření souvislých černých nebo tečkovaných ploch a k rámování textů, nazýváme grafické znaky. Pouze tyto znaky vyžadují tisk do všech šesti sloupců po dvanácti linkách. Ostatním znakům postačí generátor pro 5 sloupců po 9 linkách.

Tiskneme-li na devět linek, použijeme pro každý znak jen osm jehel. Některé znaky v řádce budeme tisknout vrchními osmi jehlami (a spodní jehla bude zahálet), jiné znaky vytiskneme spodními osmi jehlami (a vrchní jehla bude zahálet). Údaj o tom, zda se má tisknout spodními nebo vrchními jehlami uvedeme v prvku Popis_Tvaru v položce generátoru.

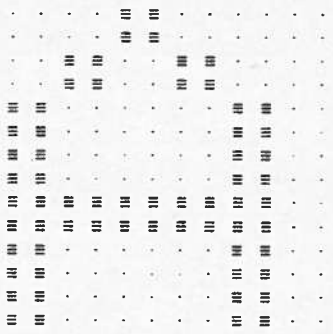
Grafické znaky mají tu vlastnost, že v 10. a 12. sloupci jen opakují obsah 6. a 8. sloupce a v nejspodnějších 4 linkách jenom opakují to, co je v nejspodnějších dvou linkách z vrchní osmice linek. Takže v 9. a 11. lince shora je totéž jako v 7. lince a v 10. a 12. lince je totéž jako v 8. lince. Určení toho, co se má tisknout ve spodních čtyřech linkách u grafických znaků, tedy můžeme klidně svěřit proceduře, která bude vyprazdňovat řádkovou paměť, takže v modulu GZ postačí, když do řádkové paměti uložíme příznak pojmenovaný RVP_12_Linek. Tím zajistíme, že k tisku grafických znaků postačí údaje o čtyřech sloupcích po osmi linkách.

Zůstává ještě otázkou, jak v generátoru odlišíme grafické znaky (které se generují zvláštním způsobem) od ostatních znaků. K tomu účelu můžeme vtipně využít údajů o prvním a posledním sloupci znaku. Tyto údaje jsou uvedeny v prvku Popis_Tvaru v položce generátoru. Jen grafické znaky totiž sahají až do dvanáctého sloupce (ostatní mají 11. a 12. sloupec prázdný). Poznamenejme, že původně byly údaje o prvním a posledním sloupci určeny pro proporcionální tisk.

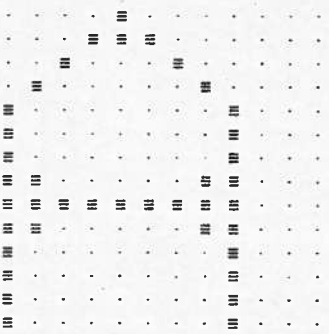
Obr. 4.3a) Obsah generátoru



Obr. 4.3b) Ledabylý tisk



Obr. 4.3c) Tučný tisk



Obr. 4.3d) Pečlivý tisk

Nyní si ještě můžeme říci několik slov o modulu RVP. Jeho úkolem je shromažďovat údaje o obsahu sloupců v jedné řádce do proměnné VP a celou řádku pak najednou vytisknout pomocí modulů OU a RU.

Jádrem modulu je proměnná VP. Je to pole složené z velkého počtu položek. Počet položek je o něco větší než největší přípustný počet sloupců v jednom řádku a každá položka obsahuje údaje potřebné k vytištění jednoho sloupce. Položka řádkové paměti VP je opět pole, složené ze dvou půlpoložek (jedna pro první průchod hlavy, druhá pro přetisk mezi linky). Každá půlpoložka je záznam složený ze dvou osmiprvkových množin, které obsahují údaje o devíti tečkách a 6 různých příznacích (mj. také RVP_12_Linek). Modul RVP umožňuje přidat novou položku do VP, vymazat z VP položky tvořící poslední znak na řádce, smazat celou VP, najet s tiskací hlavou na levý okraj. Modul dále obsahuje tři procedury pro vytištění obsahu VP.

Modul OU nezasluhuje žádné mimořádné pozornosti. Jak pro povrchní seznámení, tak i pro podrobné studium jeho funkce by měl

čtenáři stačit zápis tohoto modulu v kapitole 5.

Shrnutí:

Vysvětlili jsme pojmy záznam, pole a index. Probrali jsme moduly RVP a OU.

4.6. Další pán na holení

Tuto kapitolu bychom mohli nazvat také "kdo dřív přijde, ten dřív mele" nebo "nezavěšujte, jste v pořadí". Všechna uvedená hesla totiž vyjadřují jistý způsob čekání na obsluhu a předpokládají jisté uspořádání čekajících objektů. Takové uspořádání nazýváme

fronta.

Každý z nás zná frontu: na maso, na knihy, na zájezd s Čedokem. Fronta se vyznačuje tím, že jako první bude obsloužen ten prvek, který se první zařadil do fronty, a také další prvky budou zpracovávány v tom pořadí, ve kterém se postupně zařazovaly do fronty.

Poznamenejme, že v literatuře se ve stejném významu jako "fronta" používá i zkratka "FIFO", anglický termín "buffer" a český termín "vyrovnávací paměť", příp. "vyrovnávací paměť zpráv". Termín "fronta" se zase někdy zužuje jen na fronty procesů, s těmi se však v našem výkladu vůbec nesetkáme.

Fronta je vhodným prostředkem k předávání objektů mezi dvěma procesy. Představme si proces vaření oběda, který uskutečňuje kuchařka, a proces prodeje zeleniny, který uskutečňuje prodavačka. Kuchařka náhle zjistí, že potřebuje brambory. Vybere si ve svém okolí vhodný objekt pro funkci poskoka (dítě, manžela, tchyni apod.) a pošle jej do prodejny zeleniny. Objekt - poskok přijde do prodejny a zařadí se na konec fronty. Prodavačka vykonává proces prodeje tím způsobem, že postupně odebírá z fronty jednotlivé objekty - zákazníky v tom pořadí, ve kterém se řadili do fronty, a jednoho po druhém obsluhuje. Výše zmíněný poskok bude obsloužen tehdy, až na něj přijde řada (tj. později než všichni zákazníci, kteří přišli před ním, a dříve než všichni zákazníci, kteří přišli po něm).

Uveďme ještě jiný příklad ze života: Tuto větu můžeme považovat za frontu písmen. Čtenář četl písmena předchozí věty pravděpodobně ve stejném pořadí, v jakém je pisatel psal. Řazení písmen do knih má významnou výhodu pro spolupráci čtenářů s pisateli: pisatel klidně může zařadit do knihy další písmeno, aniž by čtenář byl musel přečíst předchozí písmeno. Pisatel řadí do knihy další a další písmena a nestará se o to, co právě dělá čtenář. Pisatele nezajímá ani to, jak rychle čtenář čte. Pisatel

si klidně píše, a čtenář se třeba ještě ani nenarodil! Také čtenář se nemusí obávat, že by mu pisatel včas nedodal další písmeno. Čtenář se nemusí starat o to, co právě dělá pisatel. Čtenář může číst knihu, aniž by pisatel musel současně psát. Čtenáře také málo zajímá, že pisatel psal po mnoho měsíců to, co on přečte za pár dní. Je tu však jedno omezení: čtenář nemůže číst to, co dosud nebylo napsáno.

Vidíme, že fronta poskytuje procesům určitou nezávislost. Proces, který frontu plní, musí dbát na to, aby nedával další objekty do plné fronty (např. čeká, až se plná fronta uvolní). A proces, který frontu vyprazdňuje, zase musí dbát na to, aby nevyjímal objekty z prázdné fronty.

Jaký užitek nám nabízejí fronty při řešení naší úlohy? Především nám umožní vyrovnat nerovnoměrné rychlosti procesů. Víme například, že přijímané znaky se napřed převádějí na množiny teček a ukládají do řádkové paměti a teprve po přijetí celého řádku se začíná tisknout. Tisk ovšem trvá zpravidla několikrát déle než příjem řádku, proto by bylo nepříjemné, kdyby byl příjem dalšího řádku pozdržen po dobu tisku předchozího řádku. Nepříjemnost odstraníme, když pokyny k provedení úkonů naskládáme do fronty, ze které si je bude vybírat ovladač mechaniky (modul OM). Frontu úkonů naprogramujeme v modulu RU.

Druhá příležitost k použití fronty se nám naskýtá při příjmu znaků. Nemůžeme předpokládat, že by nám nadřízený počítač posílal znaky právě v těch okamžicích, kdy tiskárna zrovna nemá co na práci, tedy když procedura ZZ.Zpracuj_Znak čeká na přijetí dalšího znaku. To bychom sice mohli snadno vyřešit tak, že někdy by čekala tiskárna na počítač a jindy zase počítač na tiskárnu - toho lze dosáhnout pomocí signálů na rozhraní CENTRONICS. Tím by se však zbytečně zdržovala jak tiskárna, tak i nadřízený počítač. Proto se pokusme přizpůsobit nerovnoměrnou rychlost tiskárny nerovnoměrné rychlosti nadřízeného počítače. Právě to nám totiž umožní fronta v rozhraní znaků (modul RZ), do které se budou řadit přijímané znaky.

Jak to uděláme? Od fronty vyžadujeme, aby uměla ukládat objekty do fronty a vyjímat objekty z fronty. Při vyjímání objektů z fronty je užitečné předem zjišťovat, zda není fronta prázdná. Když totiž je prázdná, může program používající frontu vykonávat jinou užitečnou činnost (a nejen nazdařbůh čekat, až se ve frontě něco objeví). Např. procedura ZZ.Zpracuj_Znak musí během čekání ošetřovat vrchní panel.

Proto budou moduly RZ a RU obsahovat po třech procedurách: jedna procedura bude ukládat do fronty, druhá bude vyjímat z fronty a třetí bude zjišťovat, zda je fronta prázdná. Prvky fronty budou uloženy v poli VP (VP je zkratka pro "vyrovnávací paměť"). Proměnná První_Plná bude obsahovat index nejstaršího prvku v poli. Proměnná První_Prázdná bude obsahovat index toho volného místa v poli, které odpovídá konci fronty.

Důvtipnému čtenáři je už asi jasné, že fronta nemusí postupovat kupředu jen tím způsobem, že první prvek fronty je z VP odstraněn a další prvky VP se posunou na místa svých předchůdců. Jednodušší je, aby prvky VP zůstaly na svých místech a aby se jen změnil obsah proměnné První_Plná. Tímto způsobem se však uvolňují položky na začátku VP "před" čelem fronty, tj. od indexu 0 do indexu První_Plná - 1. Přitom se konec pole VP postupně zaplňuje novými objekty přidávanými do fronty. Co budeme dělat, až dosáhneme konce pole VP? Řešení je nabíledni - vždyť na začátku pole VP se dříve nebo později musí objevit volné položky! Za posledním indexem pole VP tedy bude následovat znovu index 0, a tak můžeme plnit a vyprazdňovat frontu v poli VP pořád dokola.

V modulu RU je použit dosud neznámý příkaz

Dokud_platí <podmínka> **opakuj**
...
Konec {**Dokud_platí**};

Vykonávání příkazů znázorněných třemi tečkami se opakuje tak dlouho, až se změní podmínka. Je-li podmínka hned od začátku neplatná, příkaz se vůbec nevykoná a pokračuje se dalším příkazem v programu. Jestliže opakované příkazy podmínku nemění, budou se vykonávat pořád dokola (to však neplatí, může-li být podmínka změněna jiným procesem - jiný proces může opakování příkazů ukončit).

Nyní si již zvídavý čtenář může podrobně prohlédnout modul RU uvedený v kapitole 5. Modul RZ probereme později, a to v kapitole 4.7. Pečlivého čtenáře snad bude zajímat, že funkce Počet_Položek využívá operace

<celé_číslo> **mod** <kladné_číslo>.

Výsledek je zřejmý, když je levý operand kladný. Nejasnosti začínají, když je levý operand záporný. My budeme předpokládat, že se zbytek po dělení počítá stejně pro kladné i záporné levé operandy: jako rozdíl mezi levým operandem a nejbližším menším násobkem pravého operandu, např. -8 **mod** 5 = (-8) - (-10) = 2.

Bystrému čtenáři je již snad jasné, jak funguje fronta. Možná mu však vrtá v hlavě otázka, jaká nebezpečí v sobě fronta skrývá, bude-li používána dvěma procesy: jeden proces bude do fronty ukládat a druhý bude z fronty vybírat. Jeden proces bude pomocí procedury RU.Přidej_Do_VP přidávat úkony do fronty RU.VP a druhý proces bude úkony vyjímat pocedurou RU.Vyjmi_Z_VP. Proměnná RU.VP se tak stane sdílenou proměnnou, ke které bychom měli (podle kapitoly 4.2) zajistit výlučný přístup. My si však ukážeme, že fronty mají poněkud výjimečné vlastnosti.

To platí pod jednou podmínkou: v průběhu ukládání hodnot do proměnných První_Prázdná a První_Plná nesmí být procesu odebrán

procesor, tj. zmíněné proměnné musí v každém okamžiku obsahovat buď starou, anebo novou hodnotu. To bude zajištěno např. tehdy, když se nové hodnoty do zmíněných proměnných budou ukládat jedinou instrukcí procesoru.

Povězme si tedy o oněch výjimečných vlastnostech front. Procedura Přidej_Do_VP (příp. Vyjmi_Z_VP) obsahuje nebezpečné místo: vždycky napřed pracuje s prvkem pole VP, a pak teprve mění hodnotu indexu v proměnné První_Prázdná (příp. První_Plná).

Co to znamená pro proceduru Přidej_Do_VP? Představme si, že na počátku První_Prázdná = První_Plná (tj. fronta je prázdná), procedura Přidej_Do_VP napřed naplní položku VP [První_Prázdná], pak druhý proces zavolá proceduru Vyjmi_Z_VP a nakonec se změní hodnota proměnné První_Prázdná. V takovém případě procedura Vyjmi_Z_VP napřed zjistí, že fronta je prázdná (přestože je v ní již uložena správná hodnota první položky), a proto se ani nepokusí vyjmout první položku z fronty ven, nýbrž bude čekat, až se změní hodnota proměnné První_Prázdná.

Kdyby se naopak napřed měnila hodnota proměnné První_Prázdná, a pak se teprve ukládala hodnota do fronty, mohla by procedura Vyjmi_Z_VP vyjmout obsah dosud nenaplněné položky VP (protože přístup k této položce byl předčasně umožněn změnou proměnné První_Prázdná).

A co to znamená pro proceduru Vyjmi_Z_VP? Představme si, že na počátku index obsažený v proměnné První_Prázdná těsně předchází hodnotu indexu v proměnné První_Plná (tj. fronta je plná), procedura Vyjmi_Z_VP napřed vybere položku VP [První_Plná], pak druhý proces zavolá proceduru Přidej_Do_VP a nakonec se změní hodnota proměnné První_Plná. V takovém případě procedura Přidej_Do_VP napřed zjistí, že fronta je plná (přestože z ní již byla vyňata první položka), a proto se ani nepokusí přidat novou položku na konec fronty, nýbrž bude čekat, až se změní hodnota proměnné První_Plná.

Na závěr tedy můžeme říci, že procedury Přidej_Do_VP a Vyjmi_Z_VP zajišťují správné zacházení se sdílenou proměnnou VP.

Shrnutí:

Vysvětlili jsme pojem fronta a probrali jsme modul RU.

4.7. Kterak půjčovati procesor

Až dosud jsme předpokládali, že v tiskárně poběží zároveň několik procesů. Dosud však není jasné, jakým způsobem budou sdílet procesor (kromě procesu, který ošetřuje vrchní panel).

Některé programovací jazyky jsou navrženy tak, že uživateli přímo poskytují prostředky ke spouštění procesů a sdílení procesoru (nebo několika procesorů) je zajištěno bez přičinění programátora. Také systémové programy velkých počítačů zpravidla

poskytují programátorům služby potřebné pro programování souběžných procesů, takže programátoři se pak už nemusejí starat o to, jak je to uděláno.

To ale není náš případ. Řídicí počítač tiskárny je samozřejmě úplně "holý", tedy nemá žádné systémové programy a veškeré služby nám poskytuje samotné technické vybavení. Proto se nyní budeme zabývat podrobnostmi, které těsně souvisejí s technickým vybavením naší tiskárny. Poznamenejme, že člověk se může stát profesionálním programátorem i bez toho, aby do takovýchto podrobností kdy pronikl.

Jednu možnost sdílení procesoru několika procesy nám přímo poskytuje technické vybavení procesoru Z80. Jedná se o tzv. ***přerušovací systém***. Přerušovací systém, který zde zjednodušeně popíšeme, se může v podrobnostech lišit od přerušovacích systémů jiných procesorů, ale v hlavních rysech má obecnou platnost. Tato část procesoru se vztahuje k pojmu událost (INT a NMI), o kterém jsme se již zmínili v kapitole 3.4. Dojde-li k události, dokončí procesor právě rozpracovanou instrukci běžícího procesu, přeruší běh onoho procesu, ošetří událost vykonáním příslušné procedury a pak vrátí procesor původnímu procesu. Součástí přerušovacího systému jsou

```
Proměnná Přerušení_Povoleno: logický;
Procedura Povol_Přerušení;
Začátek
  Přerušení_Povoleno := pravda;
Konec {Povol_Přerušení};

Procedura Zakaž_Přerušení;
Začátek
  Přerušení_Povoleno := nepravda;
Konec {Zakaž_Přerušení};
```

Přerušovací systém zpracovává události INT jako samostatný proces takto:

```
Začátek
  Odtud_opakuj
    Platí_li Přerušení_Povoleno
        a_zároveň došlo_k_události_INT tak
      Zakaž_Přerušení;
      počkej, až se dokončí právě vykonávaná instrukce;
      INTSVC;
      vrať procesor původnímu procesu;
    Konec {Platí-li};
  až_bude_platit nepravda;
Konec;
```

Podobně jako událost INT působí na přerušovací systém i událost NMI. Proměnná Přerušení_Povoleno však nemá na přerušení NMI žádný vliv a procedura NMISVC nemůže být přerušena událostí INT. Přerušovací systém tedy zpracovává událost NMI podle návodu:

```
Proměnná Úschovna: logický;

Začátek
  Odtud_opakuj
    Platí_li došlo_k_události_NMI tak
      Úschovna := Přerušení_Povoleno;
      Zakaž_Přerušení;
      počkej, až se dokončí právě vykonávaná instrukce;
      NMISVC;
      Přerušení_Povoleno := Úschovna;
      vrať procesor původnímu procesu;
    Konec {Platí-li};
  až_bude_platit nepravda;
Konec;
```

Bystrý čtenář jistě pochopil, že přerušovací systém nám umožňuje ošetřit každou událost odpovídající procedurou. Vraťme se nyní k modulu RZ, který jsme opustili v kapitole 4.6, a pomněme, že s rozhraním znaků souvisí také událost INT (viz kapitolu 3.4). Operace Povol_Přerušení a Zakaž_Přerušení nám navíc umožňují synchronizovat proces příjmu znaků s ostatními procesy a vytvářet kritické sekce, v nichž je proces příjmu znaků vyloučen z používání jakýchkoli prostředků včetně procesoru. Příkaz Zakaž_Přerušení totiž pozdrží vykonání procedury INTSVC až do okamžiku, kdy bude (jiným procesem) vydán příkaz Povol_Přerušení. Tento mechanismus přibližně odpovídá obecnému mechanismu semaforů, který je teoreticky dobře zvládnut a zdaleka není prost jisté elegance. O semaforech se zvídavý čtenář může více dovědět z knih:

P.B.Hansen: Principy operačních systémů (SNTL, Praha, 1979),
J. Hořejš, J. Staudek, J. Brodský: Struktura počítačů a jejich programového vybavení (SNTL, Praha, 1980),
R. Pecinovský, J. Kofránek, I. Ryant: Dálkový kurs číslicové techniky, III. běh, 8. díl (Svazarm, Praha, 1987).

Poznamenejme, že k obsluze události metodou přerušení patří i uvedení procesoru do původního stavu na konci procedur INTSVC a NMISVC. Proto na začátku těchto procedur uschováme stav procesoru a na jejich konci jej obnovíme. Je třeba si také uvědomit, že před zavoláním procedury INTSVC se vždy zakáže přerušení a je úkolem procedury INTSVC buď ponechat přerušení

zakázané, anebo je na vhodném místě povolit. Takovým vhodným místem je např. poslední instrukce procedury INTSVC.

Proces příjmu znaků vykonává přerušovací systém na základě událostí INT. Snad i méně bystrý čtenář shledá jistou podobnost mezi procedurou RZ.INTSVC a RU.Přidej_Do_VP. Jestliže vyrovnávací paměť VP nebude shledána plnou, pracují zmíněné procedury úplně stejně.

Odlišným způsobem je však ošetřen případ, když je VP plná: fronta se ponechá tak, jak byla, pouze se zhasne světýlko "tiskárna připravena", obnoví se stav procesoru, ale přerušení se ponechá zakázané. Událost INT stále trvá, proces příjmu znaků je pozastaven, ale jakmile dostane příležitost (tj. jakmile se povolí přerušení), tak bude pokračovat a přerušovací systém opět přeruší. Opakované přerušení však má smysl jedině tehdy, když je ve VP volné místo. A uvolnit místo ve VP může jedině procedura RZ.Vyjmi_Z_VP. Proto se na konci této procedury rozsvítí světýlko "tiskárna připravena" a povolí se přerušení. Tím dostane proces příjmu znaků příležitost, aby přidal další znaky do fronty.

Pomocí zákazu přerušení můžeme také zabránit procesu, pomocí kterého vstupují znaky z nadřízeného počítače do tiskárny, aby používal jakýchkoli prostředků (říkáme, také, že zmíněný proces můžeme vyloučit z používání jakýchkoli prostředků). Toho využívá procedura Přetisk v modulu RVP. Když se totiž rozjede tiskací hlava, musí modul RVP rychle dodávat do fronty RU další a další pokyny k provádění úkonů. Rozjetá tiskací hlava prostě nepočká. Nadřízený počítač však o tom nic neví a mohl by plně zaměstnat procesor tiskárny událostmi INT. V tom mu však musíme zabránit - nejsnáze toho dosáhneme zákazem přerušení. Po vytištění řádky je třeba obnovit správný stav proměnné Přerušení_Povoleno v přerušovacím systému (např. podle stavu fronty RZ).

Vidíme, že mechanismus přerušení odebírá přerušovanému procesu procesor, aniž by o to přerušovaný proces požádal (a často dokonce aniž by to přerušovaný proces vůbec zjistil).

Toho využívá např. procedura ZZ.Zpracuj_Znak, která zdánlivě nesmyslně pořád dokola přemílá nesplněnou podmínku "**až_bude_platit** RZ.Je_Znak_Ve_VP". V tomto případě jsme se však lišácky spolehli na přerušovací systém, že nám nepozorovaně vloží nějaký ten znak do fronty. Podobně spoléhá na přerušení i procedura RU.Přidej_Do_VP.

Někdy se však proces potřebuje úmyslně vzdát procesoru. My se s tím setkáváme v modulu OM při časování (tj. při řízení časového rozvrhu činnosti procesu, který obsluhuje mechanickou část tiskárny). Když má např. motor posouvající hlavu udělat krok, musí být sepnuta příslušná vinutí a pak se musí poskytnout motoru čas (odměřovaný časovačem 8253), aby motor stačil krok vykonat. Po tuto dobu však nemá proces ovládání mechaniky nic na práci a naopak potřebuje, aby pracovaly ostatní procesy (aby

plnily frontu úkonů, která se nesmí vyprázdnit, dokud je hlava v pohybu).

Tuto úlohu snadno vyřešíme jiným mechanismem přidělování procesoru procesům, který můžeme označit třeba výrazem "spolupráce procesů". Ustálený název se vžil jen pro programy, které popisují spolupracující procesy:

koprogramy.

Jak už napovídá název, spolupracující procesy sdílejí procesor rovnoprávně. Nevystupují tedy jako řídící a řízený nebo věřitel (který procesor půjčuje) a dlužník (který si procesor vypůjčuje). Naopak, každý proces si ponechá procesor tak dlouho, dokud jej potřebuje, a pak se jej sám dobrovolně vzdá ve prospěch svého spolupracovníka. Procesor mu bude časem vrácen a proces bude pokračovat od místa, kde se procesoru vzdal.

To má velmi podstatnou výhodu: žádný z procesů se nemusí chovat jako procedura, která všeho všudy na začátku začíná a na konci končí (srovnej s NMISVC, INTSVC, RP.Ošetři). Spolupracující procesy se mohou vzdát procesoru hluboko ve vnořených programových strukturách a pak pokračovat dál ve své činnosti (vzdávají se tedy procesoru stejným způsobem jako program volající proceduru).

Tomu odpovídá i způsob, jakým naprogramujeme předávání procesoru. Zavedeme proceduru Transfer, která ač volána jedním procesem, vrátí se do druhého procesu (rozuměj: vrátí procesor druhému procesu). A teprve až ji zavolá druhý proces, vrátí se zpět do prvního procesu. Poznamenejme, že obecně mohou spolupracovat i více než dva procesy a že si mohou navzájem libovolně předávat procesor pomocí téže procedury Transfer. Zvídavého čtenáře snad bude zajímat, že proceduru Transfer zavedl profesor Niklaus Wirth z Curychu ve svém slavném programovacím jazyce Modula-2, popsaném např. v knize

N.Wirth: Programming in Modula-2, Springer-Verlag, Berlin Heidelberg New York Tokyo, 1985 (ruský překlad Mir, Moskva, 1987).

Procedura Transfer je deklarována:

```
Procedura Transfer (Proměnná P1, P2: Proces);
```

Procedura Transfer má dva parametry. Slovo "**Proměnná**" znamená, že se jako parametry nepředávají hodnoty dvou výrazů, nýbrž adresy dvou proměnných typu Proces. Typ Proces je takový typ, který umožňuje zachytit stav procesoru (poznamenejme, že typ Proces umožňuje uchovávat odlišné informace o stavu procesoru, než jaké se uchovávají při ošetření událostí přerušením; znalce snad bude zajímat, že každý spolupracující proces musí mít

vlastní zásobník a informace o něm je třeba uchovávat). Na adrese předané jako P2 jsou uloženy informace o procesu, kterému má být předán procesor. Do proměnné, jejíž adresa je předána parametrem P1, uloží procedura Transfer informace o předávajícím procesu. Přitom parametr P1 i P2 smí označovat tutéž proměnnou typu Proces. Proto vystačíme s jedinou proměnnou OM.Pozastavený, která bude obsahovat stav jednoho ze dvou procesů (toho, který nemá procesor).

Na začátku programu nastavíme do proměnné OM.Pozastavený požadovaný počáteční stav procesu ovládajícího mechaniku (tj. stav při spuštění procedury OM.Konej), spustíme časovač a zavoláme proceduru ZZ.Zpracuj_Znak. Tyto činnosti vykonává modul OP.

Zatímco se bude vykonávat procedura ZZ.Zpracuj_Znak, vyprší čas měřený časovačem, dojde k události NMI, zavolá se procedura NMISVC a ta (pomocí procedury Transfer) předá procesor ovladači mechaniky, jehož počáteční stav byl nastaven do proměnné OM.Pozastavený. V proměnné OM.Pozastavený se pak bude nacházet stav procesu vykonávajícího proceduru NMISVC.

Zasvěcence by snad mohlo zajímat, jak je naprogramována procedura Transfer. Je při tom využito jednak přímo jazyka procesoru 8080, jednak předpokladu, že není nutné uchovávat obsahy registrů AF, BC, DE, HL, takže postačí uchovat registr SP v proměnné OM.Pozastavený. Registr PC byl při volání procedury Transfer již uložen na zásobník, takže není třeba se o něj starat. Parametr P1 je předán do procedury v registrovém páru DE a parametr P2 v páru BC. Program je uveden v kapitole 5.

Jak tedy budeme v modulu OM programovat časování? K tomu nám poslouží procedura Perioda, která spustí časovač a pak pomocí procedury Transfer předá procesor procesu, který přijímá a zpracovává znaky. Po uplynutí nastaveného času způsobí časovač událost NMI, přerušovací systém zavolá proceduru NMISVC, která předá procesor zpět ovladači mechaniky pomocí procedury Transfer. Při dalším předání procesoru procedurou Transfer se napřed dokončí procedura INTSVC a pak se pokračuje v procesu příjmu a zpracování znaků, dokud zase nevyprší čas a nedojde k další události NMI.

Předchozí výklad by měl čtenáři postačit k tomu, aby dokázal samostatně prostudovat modul OP.

Shrnutí:

Vysvětlili jsme pojmy přerušovací systém a koprogram. Probrali jsme moduly RZ a OP.

4.8. Návrh ovladače mechaniky

Až dosud jsme při rozboru úlohy postupovali metodou shora dolů. Na ovladači mechaniky (tj. na modulu OM) si ukážeme opačný postup - zdola nahoru. Máme stanovena dvě rozhraní: rozhraní úkonů a rozhraní technických prostředků. Naším úkolem je vykonávat pokyny z rozhraní úkonů pomocí daného technického vybavení. Naši starou známou metodu shora dolů bychom použili tak, že bychom vymýšleli, jak naprogramovat jednotlivé úkony dané rozhraním úkonů.

Postup zdola nahoru je opačný: začneme od technického vybavení a budeme psát procedury, které s použitím technických prostředků nabízejí pomyslným vyšším vrstvám nějaké užitečné služby. Na základě takové nabídky služeb pak budeme navrhovat vyšší vrstvy, dokud neuspokojíme všechny požadavky určené rozhraním úkonů.

Modul OM obsahuje několik dosud neznámých příkazů. Prvním z nich je příkaz

Opakuj <výraz> **krát**
...
Konec {**Opakuj**};

Příkazy uvedené na místě tří teček se vykonají tolikrát, kolik určuje hodnota výrazu.

Tomuto příkazu se podobá příkaz, který má dvě podoby:

Pro <proměnná> := <výraz> **vzestupně_do** <výraz> **opakuj**
...
Konec {**Pro**};

nebo druhou podobu:

Pro <proměnná> := <výraz> **sestupně_do** <výraz> **opakuj**
...
Konec {**Pro**};

Tento příkaz vloží do proměnné hodnotu prvního výrazu, vykoná příkazy znázorněné třemi tečkami, pak umístí do proměnné následující nebo předchozí prvek příslušného typu, znovu vykoná příkazy znázorněné třemi tečkami - a to opakuje tak dlouho, až proměnná přesáhne hodnotu druhého výrazu. Např. "2 **vzestupně_do** 3" zopakuje příkazy znázorněné třemi tečkami dvakrát, "2 **vzestupně_do** 2" jen jednou, "2 **vzestupně_do** 1" ani jednou. Příkaz obsahující "3 **sestupně_do** 2" zopakuje příkazy znázorněné třemi tečkami dvakrát, "2 **sestupně_do** 2" jen jednou, "1 **sestupně_do** 2" ani jednou.

Dalším dosud neznámým příkazem je příkaz **Pokračuj_na_návěští**. Tento příkaz je teoretiky programování považován za velmi nebezpečný, neboť jeho vykonání narušuje strukturu programu, ve kterém je použit. Dobrý programátor se mu proto vyhýbá jako čert kříži a používá jej zcela výjimečně jen tam, kde skutečně potřebuje porušit strukturu programu. Autor jej použil jednak při zpracování řídicí posloupnosti ESC '@', kdy se opouští nedokončená procedura a pokračuje se na zcela neočekávaném místě ve volajícím programu, jednak je tento příkaz použit v modulu OM, kdy tiskací hlava se krok za krokem blíží k chybovému stavu, když tu náhle najede na levý doraz a tím k chybě nedojde.

Co je to návěští? Je to kladné celé číslo, kterým označujeme to místo v programu, na kterém má pokračovat činnost procesu po vykonání příkazu **Pokračuj_na_návěští**. Každé návěští použité v programu je napřed třeba deklarovat.

Funkce "Poř_č (<výraz>)" vrací pořadové číslo hodnoty výrazu v deklaraci odpovídajícího typu. První konstanta uvedená v deklaraci typu má pořadové číslo 0. Např. Poř_č (pravda) má hodnotu 1, Poř_č (nepravda) má hodnotu 0, Poř_č (**neplatí** pravda) má hodnotu 0, Poř_č ('A') má hodnotu 65, Poř_č ('B') má hodnotu 66, Poř_č (J1) má hodnotu 0, Poř_č (J8) má hodnotu 7.

Opačná funkce se označuje jménem typu. Tak např. logický (0) má hodnotu nepravda, znak (66) má hodnotu 'B', HW.Signál (6) má hodnotu S6, HW.Jehla (7) má hodnotu J8.

Významné rozšíření našeho výkladu o složitě strukturovaných typech představuje použití deklarace záznamu ve tvaru:

```
záznam
  Vyber <typ> ze
  <konstanta_1>: ( <seznam položek záznamu> );
    ...
  <konstanta_n>: ( <seznam položek záznamu> );
Konec;
```

Tento tvar deklarace záznamu najdeme např. u typu RU.Popis_Úkonu. Potřebujeme totiž, aby některá položka obsahovala počet kroků, které má vykonat některý motor, jiná položka aby obsahovala označení písničky, kterou je třeba zahrát, jiná položka zase musí obsahovat kombinaci jehel, které se mají otisknout na papír. Všechny položky však musí být stejného typu, aby mohly být uspořádány do pole RU.VP. Toho lze dosáhnout jednak tím způsobem, že by každá položka typu Popis_Úkonu obsahovala prvky všech potřebných typů (a využila z nich vždy jenom jeden), jednak tím způsobem že do položek typu Popis_Úkonu dovolíme ukládat data několika různých typů, které vyjmenujeme v deklaraci a při každém konkrétním použití položky typu Popis_Úkonu pak musíme zvolit správný typ prvku.

Stejného účinku lze dosáhnout i deklarací položky jen pro jediný typ dat a při použití položek pak zvolit požadovaný typ přetypováním (příkazem **Typ**). Přetypování však v sobě skrývá nebezpečí, že položku omylem přetypujeme na nesmyslný typ, zatímco vyjmenováním přípustných typů v deklaraci záznamu si zajistíme alespoň jakous takous kontrolu.

V modulu OM se vyskytuje volání několika procedur, které nejsou nikde deklarovány. Autor zvolil tento způsob zápisu pro takové příkazy, které bude třeba napsat v jazyce použitého procesoru. Jedná se o procedury Vykonej_Krok a Vystup_Na_Port. Tyto procedury nahradí např. v jazyce procesoru 8080 instrukce OUT, v jazyce procesoru Z80 instrukce OUTC. Význam procedur by měl být zřejmý a jejich realizace přesahuje rámec našeho výkladu.

Nyní můžeme přistoupit k výkladu jednotlivých procedur.

Jednu ze základních služeb poskytuje procedura Zahraj_Písničku. Součástí procedury Zahraj_Písničku je tabulka všech nabízených písniček nazvaná "Písnička". Je to vlastně pole, jehož indexy jsou pojmenovány významem jednotlivých písniček (indexy jsou typu RU.Význam_Písničky). Každý prvek pole "Písnička" obsahuje popis jedné písničky. Každá písnička se skládá ze 4 tónů různé výšky a délky. Každý tón je popsán záznamem, ve kterém je uvedena jednak výška tónu, jednak délka tónu. Výška tónu je celé číslo, které udává dobu jednoho kmitu v mikrosekundách. Délka tónu je celé číslo, které udává dobu trvání tónu v milisekundách. Procedura Zahraj_Písničku vybere z tabulky položku určenou parametrem P a postupně rozezní pípák čtyřmi tóny zvolené písničky.

Služby Zahraj_Písničku využívá procedura Ohlas_Chybu, která jednak ohlásí chybu na rozhraní CENTRONICS (tím podá zprávu nadřízenému počítači), jednak zahraje příslušnou písničku (tím podá zprávu obsluze).

Další významnou službu poskytuje procedura Najeď_Doleva. Po zapnutí tiskárny se hlava nachází na neznámém místě a je třeba najet s ní na levý doraz. Před tím, než začneme pohybovat hlavou, je užitečné zkontrolovat napájení motorů. Pokud nejsou motory napájeny, ohlásí se chyba. Jsou-li motory v pořádku a nachází-li se hlava na levém dorazu, posune se hlava mimo levý doraz (mohla by se nacházet příliš vlevo). Potom se pokoušíme najet hlavou zprava na levý doraz. Víme, že i z nejzazší pravé polohy dosáhneme levého dorazu menším počtem kroků než např. 1320. Může se však stát, že se hlava z nějakého důvodu zasekne nebo že nebude fungovat čidlo levého dorazu - v takovém případě se nám ani po vykonání 1320 kroků nepodaří dosáhnout levého dorazu. Proto i v tomto případě ohlásíme chybu. Podaří-li se správně najet hlavou na levý doraz, vykoná se ještě posun papíru o jeden krok vzhůru, aby se v prvním řádku zbytečně neprojevila vůle v převodech válce.

Důležité služby poskytují také procedury Rozjeď a Brzdi. Procedura Rozjeď rozjede hlavu požadovaným směrem (parametr Směr) na požadovanou rychlost (v proměnné Perioda_Hlavy). Procedura Brzdi zabrzdí rozjetou hlavu.

Procedura Vodorovně vykonává veškeré činnosti hlavy. V případě, že tiskne, přijímá kombinace jehel z rozhraní úkonů.

Nejvyšší vrstvu v modulu OM představuje procedura Konej, která s využitím nabízených služeb vykonává úkony podle požadavků přijímaných z rozhraní úkonů.

Shrnutí:

Probrali jsme modul OM, který je vytvořen metodou zdola nahoru.

4.9. Odhalení skutečnosti

Mnohý čtenář pod vlivem vlastního zdravého rozumu asi nevěří, že by výše předvedený způsob rozboru úlohy mohl vést tak přímočaře k výsledkům tak oslnivým. Nuže nalijme si čistého vína. Především je třeba říci, že přímá cesta od zadání úlohy k návrhu programu skutečně existuje (byla zde předvedena). Současně je však třeba dodat: naděje, že se podaří nalézt tuto cestu hned na první pokus, je pramalá.

A skutečně! Rozbor úlohy je bloudění neznámou krajinou. Někdy nás sveze laskavý povozník v podobě programovacího jazyka vyšší úrovně, jindy se musíme prodírat houštinami instrukcí zvoleného procesoru. Někdy lze použít hotových modulů, někdy se musíme uchýlit k programátorské fintě. Bloudíme po cestách, které nikam nevedou, a jen občas se podaří proniknout k dílčímu řešení. A když se konečně dostaneme až k cíli, obvykle zjistíme, že jsme ani zdaleka nešli tou nejlepší cestou, program je třeba moc pomalý nebo se nevejde do paměti. Pak musíme hledat dál lepší a lepší cesty tak dlouho, až nás některá jakž takž uspokojí. A víme, že to sice není nejlepší cesta, ale další hledání by se prostě už nevyplatilo.

Rozbor úlohy by se však neměl stát bezcílným blouděním v temnotách. Vždy bychom měli znát, odkud a kam se potřebujeme dostat. To znamená, že napřed potřebujeme znát vrchní a spodní rozhraní té vrstvy, kterou navrhujeme, a teprve potom můžeme vyplňovat prostor mezi rozhraními - tj. hledat cestu od vrchního rozhraní ke spodnímu nebo naopak.

Krkolomnými cestami se ubíral i autor Veselé rukověti při psaní a zkoušení programu pro řízení tiskárny. Napřed vytvořil jen velmi zjednodušený ovladač mechaniky, kterému zadával pomocí nadřízeného počítače pokyny k vykonávání jednotlivých kroků s motory a ke hraní písniček. Na základě předběžného návrhu vyšších vrstev pak upravil rozhraní úkonů přibližně do současné

podoby. Postupně byl program rozšířen o ošetřování vrchního panelu a o rozjezdy a brzdění motoru posouvajícího hlavu. Rozhraní znaků bylo umístěno v nadřízeném počítači a zpočátku přijímalo znaky z klávesnice, později četlo znaky z disku (aby bylo možné tisknout předem připravené texty).

Došlo i k jednomu velmi podstatnému zásahu do struktury programu. Zpočátku se kódy znaků přímo ukládaly do řádkové paměti a znaky se generovaly až do rozhraní úkonů. Jisté potíže však nastaly s tiskem obrázků, a proto se autor nakonec rozhodl pro generování znaků již do řádkové paměti. Tato zásadní změna struktury však pro autora znamená natolik trpkou zkušenost, že autor vřele doporučuje všem čtenářům, aby neměnili navrženou strukturu programu. A podotýká, že jediná možnost, jak toho dosáhnout, spočívá v pečlivém studiu zadání úlohy a v důkladném návrhu struktury programu.

Autor doufá, že těmito stručnými poznámkami učinil zadost čtenářově touze po pravdě o rozboru úlohy. Čtenář jistě chápe, že ve výkladu nebylo možné přesně sledovat skutečný postup rozboru úlohy. Proto autor zvolil postup "tak jak by se to bylo bývalo mělo správně udělat", který je jednak stručnější, jednak srozumitelnější.

Shrnutí:

V kapitole 4 jsme rozebrali zadanou úlohu. Vysvětlili jsme pojmy modul, procedura, proces, sdílený prostředek, výlučný přístup, kritická sekce, pole, záznam, fronta, přerušení a koprogramy. Seznámili jsme se se strukturovanými daty a se základními programovými konstrukcemi. Navrhli jsme modulární strukturu programu, jednotlivé procesy, rozhraní mezi procesy a způsoby sdílení prostředků mezi procesy. Zvláštní pozornost jsme věnovali sdílení procesoru.

5. Navržené programy

Tato kapitola přináší zápis programů, který je výsledkem našeho rozboru úlohy. Jsou zde uvedeny veškeré podrobnosti k výkladu z kapitoly 4. Studium této kapitoly není možné bez znalosti kapitoly 4. Uvedené programy mají sloužit jako příklady k předchozímu výkladu a v případě potřeby osvětlit nejasná místa. Zvídavý čtenář zde najde podrobnosti o řešení úlohy. Smysl kapitoly je však především ilustrativní, takže většina čtenářů může podrobné studium této kapitoly vynechat.

V uvedených programech se vyskytují některé dosud nevysvětlené obraty.

Autor si dovoluje upozornit čtenáře, že nejasná jazyková konstrukce "**Platí_li neplatí** <podmínka>" znamená "Neplatí-li <podmínka>" a podobně "**Dokud_platí neplatí** <podmínka>" znamená "Dokud neplatí <podmínka>". Operátor "**neplatí**" prostě obrací význam podmínky (z hodnoty pravda na hodnotu nepravda a z hodnoty nepravda na hodnotu pravda).

V modulu OU se vyskytuje funkce "abs (<výraz>)". Výraz musí být celočíselného typu a výsledkem funkce je absolutní hodnota výrazu. To znamená, že funkce abs odstraňuje znaménko minus ze záporných čísel. Např. abs (-1) má hodnotu 1, abs (3) má hodnotu 3.

Další dosud nevysvětlený jev obsahuje procedura Udělej_Přetisk. Jeden z jejích parametrů je označen slovem "**Procedura**". Tento parametr se skutečně uvnitř procedury Udělej_Přetisk používá jako procedura. Tím je umožněno proceduře Tisk, která volá proceduru Udělej_Přetisk, aby určila, zda se má tisknout procedurou Tisk_Sloupce_9 (která tiskne linky 1 až 9), anebo procedurou Tisk_Sloupce_12 (která tiskne linky 9 až 12). Procedura Tisk předá proceduře Udělej_Přetisk jednu ze dvou zmíněných procedur jako parametr a procedura Udělej_Přetisk pak voláním parametru Tisk_Sloupce automaticky volá tu proceduru, která jí byla předána jako parametr.

Nyní již může zvídavý čtenář přistoupit k podrobnému studiu uvedených programů.

```
Modul ZZ;
{ Zpracování znaků }
{ Oprava z 09.06.88/14:15 }

Vývoz
  Zpracuj_Znak;

Dovoz
  HW, RP, RU, OU, RVP, GZ, GZTAB;

Typ
  Národ = (USA, Francie, Německo, Británie, Dánsko_I,
           Švédsko, Itálie, Španělsko_I, Japonsko,
           Norsko, Dánsko_II, Španělsko_II, Lat_Amerika);
```

```
Proměnná
 { Proměnné pro vodorovný posun }
  Hustota_KLYZ: pole [09h..0Ch] ze bajt; { mapuj i mod 16 }
  { !!! hodnota proměnné Hustota_Grafiky !!! }
  { !!! bude předem pevně nastavena      !!! }
  Hustota_Grafiky: pole [0..7] ze záznam
    Skutečná,
    Požadovaná: bajt;
  Konec {Hustota_Grafiky};

 { Proměnné pro svislý posun }
  Předvolba_Řádkování: celý; { v 1/432" pro ESC A a ESC 2 }

 { Různé příznaky }
  Nastav_Bit_8,              { maska pro 7 bitový příjem }
  Zruš_Bit_8: bajt;          { maska pro 7 bitový příjem }
  Auto_LF,                   { samovolné řádkování při CR }
  IBM,                       { poplatnost zvyklostem IBM }
  Bezprostřední_Tisk,        { přijatý znak se hned tiskne }
  Řídicí_Taky_Přes_80h,      { zapnuta druhá sada říd. znaků }
  NUL_CAN_Řídicí: logický; { kódy 0..01Fh se netisknou }

{*********************************************************}

Procedura Zpracuj_Znak (Předvolba: HW.Předvolba);
{ Opakovaně čte znaky z rozhraní znaků, opakovaně půjčuje }
{ procesor rozhraní panelu, zpracovává řídicí znaky a po- }
{ sloupnosti, tiskne obrázky i tištitelné znaky pomocí mo-}
{ dulů GZ, RVP, OU a RU.                                  }
Návěští 1;
Proměnná
  Přečtený_Znak: bajt;

{=========================================================}

Procedura Čti_Znak;
{ Naplní proměnnou Přečtený_Znak z RZ a ošetří 8. bit  }
{ !!! Smí se volat jen když platí RZ.Je_Znak_Ve_VP !!! }
Začátek
  Přečtený_Znak := RZ.Vyjmi_Z_VP;
  Přečtený_Znak Typ Osmice_Signálů :=
    (Přečtený_Znak Typ Osmice_Signálů +
    Nastav_Bit_8 Typ Osmice_Signálů) *
    Zruš_Bit_8 Typ Osmice_Signálů;
Konec {Čti_Znak};

{*********************************************************}

Funkce Čti_Další_Znak: bajt;
Začátek
  Odtud_opakuj
  až_bude_platit RZ.Je_Znak_Ve_VP;
  Čti_Znak; { !vedlejší účinek na proměnnou Přečtený_Znak! }
  Čti_Další_Znak := Přečtený_Znak;
Konec {Čti_Další_Znak};

{*********************************************************}
```

```
Funkce Čti_Číslo: celý;
{ přečte 2 bajty z rozhraní znaků a vytvoří z nich číslo }
Proměnná
  Nižší_Řády: bajt;
Začátek
  Nižší_Řády := Čti_Další_Znak;
  Čti_Číslo := Nižší_Řády + 256 * Čti_Další_Znak;
Konec {Čti_Číslo};

{************************************************************}

Procedura Hustilka (Hustota: bajt; Výška_9: logický);
{ tiskne obrázek ve zvolené hustotě na 8 nebo 9 linek }
Proměnná
  i,
  Skutečný_Krok,
  Požadovaný_Krok,
  Skutečná_Poloha,
  Požadovaná_Poloha: celý;
  Sloupec: záznam Spodní_Linka, Vrchních_8: bajt; Konec;
  Položka: RVP.Položka;
  { !!! hodnota proměnné Prázdná_Položka !!! }
  { !!! bude předem pevně nastavena      !!! }
  Prázdná_Položka: RVP.Položka;
Začátek {Hustilka}
  { !!! hodnota proměnné Prázdná_Položka !!! }
  { !!! bude předem pevně nastavena:     !!! }
    Prázdná_Položka [RVP.Základ].Spodek := [];
    Prázdná_Položka [RVP.Základ].Vršek := [];
    Prázdná_Položka [RVP.Přetisk].Spodek := [];
    Prázdná_Položka [RVP.Přetisk].Vršek := [];
  Položka [RVP.Základ].Spodek := [];
  Položka [RVP.Základ].Vršek := [RVP.Hranice_Znaku];
  Položka [RVP.Přetisk].Spodek := [];
  Položka [RVP.Přetisk].Vršek := [];
  Skutečná_Poloha := 0;
  Požadovaná_Poloha := 0;
  Skutečný_Krok := Hustota_Grafiky [Hustota].Skutečná;
  Požadovaný_Krok := Hustota_Grafiky [Hustota].Požadovaná;
  Pro i := 1 vzestupně_do Čti_Číslo opakuj
    Sloupec.Vrchních_8 := Čti_Další_Znak;
    Platí_li Výška_9 tak
      Sloupec.Spodní_Linka := Čti_Další_Znak;
    jinak
      Sloupec.Spodní_Linka Typ Osmice_Jehel := [];
    Konec {Platí_li};
    { dělení 128 se provede posunem o 7 bitů doprava }
    Sloupec.Spodní_Linka Typ celý :=
      Sloupec.Spodní_Linka Typ celý děl 128;
    Položka [RVP.Základ].Spodek :=
      Položka [RVP.Základ].Spodek
      + Sloupec.Spodní_Linka Typ Osmice_Jehel;
    Položka [RVP.Základ].Vršek :=
      Položka [RVP.Základ].Vršek
      + Sloupec.Vrchních_8 Typ RVP.Popis_Vršku;
    Platí_li Skutečná_Poloha <= Požadovaná_Poloha tak
      RVP.Přidej_Sloupec (Položka);
      Platí_li Skutečný_Krok > 12 tak
        RVP.Přidej_Sloupec (Prázdná_Položka);
```

```
      Konec {Platí_li};
      Skutečná_Poloha := Skutečná_Poloha + Skutečný_Krok;
      Položka [RVP.Základ].Spodek := [];
      Položka [RVP.Základ].Vršek := [];
    Konec {Platí_li};
    Požadovaná_Poloha := Požadovaná_Poloha + Požadovaný_Krok;
  Konec {Pro};
  Platí_li RVP.První_Prázdná > RVP.Pravý_Okraj + 1 tak
    RVP.První_Prázdná := RVP.Pravý_Okraj + 1;
  Konec {Platí_li};
Konec {Hustilka};

{*********************************************************}

Procedura Naplň_Tvar_Znaku (x: bajt);
{ Přijímá bajty z rozhraní znaků a           }
{ naplní jimi GZ.Gen [GZ.RWM, x]. Tvar_Znaku }
Proměnná
  j: bajt;
Začátek
  Pro j := 0 vzestupně_do 4 opakuj
    GZ.Gen [GZ.RWM, x].Tvar_Znaku [j] Typ bajt :=
      Čti_Další_Znak;
    Přečtený_Znak := Čti_Další_Znak; { vynech }
  Konec {Pro};
  Přečtený_Znak := Čti_Další_Znak; { vynech }
Konec {Naplň_Tvar_Znaku};

{*********************************************************}

Procedura Najeď_Na (x: celý);
{ Zajistí, aby se další znak tiskl od polohy x/60*25,4 mm. }
Začátek
  Platí_li (RVP.Levý_Okraj <= x) a_zároveň
      (x <= RVP.Pravý_Okraj) tak
    RVP.Tiskni_Kus_Řádku;
    RVP.První_Prázdná := x;
    RVP.První_Plná := x;
  Konec {Platí_li};
Konec {Najeď_Na};

{*********************************************************}

Funkce Řídicí (x: bajt): logický;
{ Zjistí, zda se má znak x zpracovat jako řídicí znak }
Začátek
  Platí_li Řídicí_Taky_Přes_80h tak
    x := x mod 80h;
  Konec {Platí_li};
  Řídicí := (x < 020h) a_zároveň (NUL_CAN_Řídicí nebo
    ((x <> 000h {NUL}) a_zároveň (x <> 018h {CAN})));
Konec {Řídicí};

{*********************************************************}

Procedura Naplň_Svislý_Tabulátor (x: bajt);
{ Nastaví nové zarážky do kanálu x svislého tabulátoru }
Proměnná
  i: bajt;
```

```
Začátek
  OU.Smaž_Tabulátory;
  i := 1;
  Dokud_platí Čti_Další_Znak <> 0 opakuj
    Platí_li i <= OU.Zarážek_V_Kanálu tak
      OU.Svislé_Tabulátory [x, i] :=
        Přečtený_Znak * OU.Rozteč_Řádků;
      i := i + 1;
    Konec {Platí_li};
  Konec {Dokud_platí};
Konec {Naplň_Svislý_Tabulátor};

{************************************************************}

Funkce Šířka_Mezery: celý;
{ Zjistí šířku znaku mezera ve 1/120 * 25,4 mm podle }
{ druhu písma.                                       }
Začátek
  Šířka_Mezery := GZ.Prostrkání +
    (1 + poř_č
      (GZ.Široké_Do_Konce_Řádku nebo GZ.Široké_Do_Vypnutí))
    * ((2 - poř_č (GZ.Zhuštěný)) *
    (6 + poř_č (GZ.Zhuštěný) - poř_č (GZ.Elite)));
Konec {Šířka_Mezery};

{************************************************************}

Funkce Omez (x: celý): celý;
{ Přepočte vzdálenost okraje x mezer zleva na 1/120*25,4 mm }
{ a omezí tuto hodnotu, aby nepřesáhla pravý doraz.         }
Začátek
  x := x * Šířka_Mezery - 1;
  Platí_li x <= RVP.Pravý_Doraz tak
    Omez := x;
  jinak
    Omez := RVP.Pravý_Doraz;
  Konec {Platí_li};
Konec {Omez};

{************************************************************}

Procedura Nastav_Okraje (Levý, Pravý: celý);
{ !!! Levý i Pravý <= RVP.Pravý_Doraz !!! }
Začátek
  Platí_li (Levý+23) > Pravý {aspoň 1 znak rozšíř. pica} tak
    RU.Přidej_Do_VP (OU.Nezdar);
  jinak
    RVP.Levý_Okraj := Levý;
    RVP.Pravý_Okraj := Pravý;
    RVP.Tiskni_A_Doleva;
    GZ.Smaž_Tabulátory;
  Konec {Platí_li};
Konec {Nastav_Okraje};

{************************************************************}
```

```
Procedura Zpracuj_ESC;
{ Zpracuje řídicí posloupnost za ESC }
Proměnná
  x: celý;
  i: bajt;
  s: Osmice_Signálů;
Začátek
  Vyber Čti_Další_Znak ze
  00Ah: { ESC LF }
    Začátek
      RVP.Tiskni_Kus_Řádku;
      Platí_li OU.Poloha_Papíru <= OU.Rozteč_Řádků tak
        OU.Posuň_Papír_Na (0);
      jinak
        OU.Posuň_Papír_Na
          (OU.Poloha_Papíru - OU.Rozteč_Řádků);
      Konec {Platí_li};
    Konec;
  00Ch: { ESC FF }
    Začátek
      RVP.Tiskni_Kus_Řádku;
      OU.Posuň_Papír_Na (0);
    Konec;
  00Eh: { ESC SO }
    Začátek
      GZ.Široké_Do_Konce_Řádku := pravda;
    Konec;
  00Fh: { ESC SI }
    Začátek
      GZ.Zhuštěný := pravda;
    Konec;
  020h: { ESC SP }
    Začátek
      GZ.Prostrkání := Čti_Další_Znak;
    Konec;
  021h: { ESC '!' }
    Začátek
      s Typ bajt := Čti_Další_Znak;
      GZ.Šlité              := S0 je_prvkem s;
      GZ.Proporcionální     := S1 je_prvkem s;
      GZ.Zhuštěný           := S2 je_prvkem s;
      GZ.Mezi_Sloupce       := S3 je_prvkem s;
      GZ.Příznaky := GZ.Příznak (poř_č (S4 je_prvkem s));
      GZ.Široké_Do_Vypnutí := S5 je_prvkem s;
      GZ.Podtržený          := S7 je_prvkem s;
    Konec;
  023h: { ESC '#' }
    Začátek
      { osmý bit přijímaných znaků bude ponechávána }
      Nastav_Bit_8 := 0;
      Zruš_Bit_8 := 0FFh;
    Konec;
  024h: { ESC '$' }
    Začátek
      Najeď_Na (RVP.Levý_Okraj + (2 * Čti_Číslo));
    Konec;
```

```
025h: { ESC '%' }
  Začátek
    { výběr ROM nebo RWM generátoru }
    GZ.Vybraný := GZ.Druh (Čti_Další_Znak mod 2);
  Konec;
026h: { ESC '&' }
  Začátek
    i := Čti_Další_Znak; { nevšímej si ho }
    i := Čti_Další_Znak; { první kód k přepsání }
    Pro i := i vzestupně_do Čti_Další_Znak opakuj
      GZ.Gen [GZ.RWM, i].Popis_Tvaru := Čti_Další_Znak;
      Naplň_Tvar_Znaku (i);
    Konec {Pro};
  Konec;
02Ah: { ESC '*' }
  Začátek
    Hustilka (Čti_Další_Znak mod 8, nepravda)
  Konec;
02Dh: { ESC '-' }
  Začátek
    GZ.Podtržený Typ bajt := Čti_Další_Znak mod 2;
  Konec;
02Fh: { ESC '/' }
  Začátek
    OU.Kanál_Svislé_Tabulace := Čti_Další_Znak mod 8;
  Konec;
030h: { ESC '0' }
  Začátek
    OU.Rozteč_Řádků := 432 děl 8;
  Konec;
031h: { ESC '1' }
  Začátek
    OU.Rozteč_Řádků := 7 * (432 děl 72);
  Konec;
032h: { ESC '2' }
  Začátek
    Platí_li IBM tak
      OU.Rozteč_Řádků := Předvolba_Řádkování;
    jinak
      OU.Rozteč_Řádků := 432 děl 6;
    Konec {Platí_li};
  Konec;
033h: { ESC '3' }
  Začátek
    OU.Rozteč_Řádků := Čti_Další_Znak * (432 děl 216);
  Konec;
034h: { ESC '4' }
  Začátek
    Platí_li IBM tak
      OU.Poloha_Papíru := 0;
    jinak
      RU.Přidej_Do_VP (OU.Nezdar)
    Konec {Platí_li};
  Konec;
```

```
035h: { ESC '5' }
  Začátek
    Platí_li IBM tak
      Auto_LF Typ bajt := Čti_Další_Znak mod 2;
    jinak
      RU.Přidej_Do_VP (OU.Nezdar);
    Konec {Platí_li};
  Konec;
036h, 037h: { ESC '6', ESC '7' }
  Začátek
    Řídicí_Taky_Přes_80h Typ bajt := Přečtený_Znak mod 2;
  Konec;
038h, 039h: { ESC '8', ESC '9' }
  Začátek
    RP.Kontroluj_Papír Typ bajt := Přečtený_Znak mod 2;
  Konec;
03Ah: { ESC ':' }
  Začátek
    Platí_li IBM tak
      GZ.Élite := pravda;
    jinak
      Opakuj 3 krát
        Platí_li Čti_Další_Znak <> 0 tak
          RU.Přidej_Do_VP (OU.Nezdar);
        Konec {Platí_li};
      Konec {Opakuj};
      GZ.Gen [GZ.RWM] := GZ.Gen [GZ.ROM];
    Konec {Platí_li};
  Konec;
03Ch: { ESC '<' }
  Začátek
    RVP.Tiskni_A_Doleva;
    RVP.Hlavu_Doleva;
  Konec;
```

```
03Dh: { ESC '=' }
  Začátek
    Platí_li IBM tak
      x := Čti_Číslo;
      Platí_li Čti_Další_Znak <> 020h tak
        RU.Přidej_Do_VP (OU.Nezdar);
      jinak
        i := Čti_Další_Znak;
        Dokud_platí x > 0 opakuj
          GZ.Gen [GZ.RWM, i].Popis_Tvaru := Čti_Další_Znak;
          Platí_li Čti_Další_Znak <> 0 tak
            RU.Přidej_Do_VP (OU.Nezdar);
            i := 0;
          jinak
            Naplň_Tvar_Znaku (i);
            x := x - 13;
            i := i + 1;
          Konec {Platí_li};
        Konec {Dokud_platí};
      Konec {Platí_li}
    jinak { EPSON }
      { u přijímaných znaků bude osmý bit nastavován na 0 }
      Nastav_Bit_8 := 0;
      Zruš_Bit_8 := 07Fh;
    Konec {Platí_li};
  Konec;
03Eh: { ESC '>' }
  Začátek
    { u přijímaných znaků bude osmý bit nastavován na 1 }
    Nastav_Bit_8 := 080h;
    Zruš_Bit_8 := 0FFh;
  Konec;
03Fh: { ESC '?' }
  Začátek
    i := Čti_Další_Znak;
    Hustota_KLYZ [i mod 16] := Čti_Další_Znak mod 8;
  Konec;
040h: { ESC '@' }
  Začátek
    RVP.Hlavu_Doleva; { aby se mohlo volat RVP.Nastav }
    Pokračuj_na_návěští 1;
  Konec;
041h: { ESC 'A' }
  Začátek
    Předvolba_Řádkování := Čti_Další_Znak * (432 děl 72);
    Platí_li neplatí IBM tak
      OU.Rozteč_Řádků := Předvolba_Řádkování;
    Konec {Platí_li};
  Konec;
042h: { ESC 'B' }
  Začátek
    Naplň_Svislý_Tabulátor (0);
  Konec;
```

```
043h: { ESC 'C' }
  Začátek
    Platí_li Čti_Další_Znak = 0 tak
      OU.Výška_Stránky := Čti_Další_Znak * (432 + 6);
    jinak
      OU.Výška_Stránky := Přečtený_Znak * OU.Rozteč_Řádků;
    Konec {Platí_li};
  Konec;
044h: { ESC 'D' }
  Začátek
    GZ.Smaž_Tabulátory;
    i := 1;
    Dokud_platí Čti_Další_Znak <> 0 opakuj
      Platí_li i <= GZ.Počet_Zarážek tak
        GZ.Vodorovné_Tabulátory [i] :=
          Low (Přečtený_Znak * Šířka_Mezery);
        i := i + 1;
      Konec {Platí_li};
    Konec {Dokud_platí};
  Konec;
045h: { ESC 'E' }
  Začátek
    GZ.Mezi_Sloupce := pravda;
  Konec;
046h: { ESC 'F' }
  Začátek
    GZ.Mezi_Sloupce := nepravda;
  Konec;
047h: { ESC 'G' }
  Začátek
    GZ.Příznaky := GZ.Mezi_Linky;
  Konec;
049h: { ESC 'I' }
  Začátek
    s Typ bajt := Čti_Další_Znak;
    Platí_li IBM tak
      GZ.Vybraný := GZ.Druh (poř_č (S2 je_prvkem s));
      GZ.Pečlivý := S1 je_prvkem s;
    jinak { EPSON }
      NUL_CAN_Řídicí := neplatí (S0 je_prvkem s);
      Řídicí_Taky_Přes_80h := NUL_CAN_Řídicí;
    Konec {Platí_li};
  Konec;
04Ah: { ESC 'J' }
  Začátek
    RVP.Tiskni_Kus_Řádku;
    OU.Posuň_Papír_Na
      (OU.Poloha_Papíru + Čti_Další_Znak * (432 děl 216));
  Konec;
04Bh, 04Ch, 059h, 05Ah:
    { ESC 'K', ESC 'L', ESC 'Y', ESC 'Z' }
  Začátek
    Hustilka (Hustota_KLYZ [Přečtený_Znak mod 16],
      nepravda);
  Konec;
04Dh: { ESC 'M' }
  Začátek
    GZ.Élite := pravda;
  Konec;
```

```
04Eh: { ESC 'N' }
  Začátek
    OU.Přeskok_Perforace :=
      Čti_Další_Znak * OU.Rozteč_Řádků;
  Konec;
04Fh: { ESC 'O' }
  Začátek
    OU.Přeskok_Perforace := 0;
  Konec;
050h: { ESC 'P' }
  Začátek
    GZ.Elite := nepravda;
  Konec;
051h: { ESC 'Q' }
  Začátek
    i := Čti_Další_Znak; { podle IBM i EPSON chce další }
    Platí_li IBM tak
      RU.Přidej_Do_VP (OU.Nezdar);
    jinak
      Nastav_Okraje (RVP.Levý_Okraj, Omez (i));
    Konec {Platí_li};
  Konec;
052h: { ESC 'R' }
  Začátek
    Platí_li IBM tak
      OU.Smaž_Tabulátory;
      GZ.Plň_Tabulátory;
    jinak
      GZ.Gen [GZ.ROM] := GZTAB.IBM; { nastaví sadu USA }
      Platí_li Národ (Čti_Další_Znak) > Lat_Amerika tak
        { čeština KOI-8čs }
        GZ.Česky;
      jinak_platí_li Národ (Přečtený_Znak) > USA tak
        { Francie..Lat_Amerika }
        Pro i := GZTAB.
            Národní_Sada [Národ (Přečtený_Znak)].První
            vzestupně_do GZTAB.
            Národní_Sada [Národ (Přečtený_Znak)].Poslední
            opakuj
          GZ.Gen [GZ.ROM, GZTAB.Přeskupení [i].Kam]
            := GZTAB.IBM [GZTAB.Přeskupení [i].Odkud];
        Konec {Pro};
      Konec {Platí_li};
      { pro USA zůstane GZTAB.IBM beze změny }
    Konec {Platí_li};
  Konec;
053h: { ESC 'S' }
  Začátek
    GZ.Příznaky := GZ.Příznak (poř_č (GZ.Horní_Index)
      + Čti_Další_Znak mod 2);
  Konec;
054h, 048h: { ESC 'T', ESC 'H' }
  Začátek
    GZ.Příznaky := GZ.Základní;
  Konec;
055h: { ESC 'U' }
  Začátek
    RVP.Zleva_Doprava Typ bajt := Čti_Další_Znak mod 2;
  Konec;
```

```
057h: { ESC 'W' }
  Začátek
    GZ.Široké_Do_Vypnutí Typ bajt := Čti_Další_Znak mod 2;
  Konec;
058h: { ESC 'X' }
  Začátek
    i := Čti_Další_Znak;
    Nastav_Okraje (Omez (i), Omez (Čti_Další_Znak));
  Konec;
05Ch: { ESC '\' }
  Začátek
    Platí_li IBM tak
      Pro x := 1 vzestupně_do Čti_Číslo opakuj
        GZ.Zpracuj_Znak (Čti_Další_Znak);
      Konec {Pro};
    jinak
      Najeď_Na (RVP.První_Prázdná + (2 * Čti_Číslo));
    Konec {Platí_li};
  Konec;
05Eh: { ESC '^' }
  Začátek
    Platí_li IBM tak
      GZ.Zpracuj_Znak (Čti_Další_Znak);
    jinak
      Hustilka (Čti_Další_Znak mod 8, pravda);
    Konec {Platí_li};
  Konec;
05Fh: { ESC '_' }
  Začátek
    GZ.Nadtržený Typ bajt := Čti_Další_Znak mod 2;
  Konec;
061h: { ESC 'a' }
  Začátek
    RVP.Zarovnání := Čti_Další_Znak mod 4;
  Konec;
062h: { ESC 'b' }
  Začátek
    Naplň_Svislý_Tabulátor (Čti_Další_Znak);
  Konec;
069h: { ESC 'i' }
  Začátek
    Bezprostřední_Tisk Typ bajt := Čti_Další_Znak mod 2;
  Konec;
06Ah: { ESC 'j' }
  Začátek
    RVP.Tiskni_Kus_Řádku;
    x := OU.Poloha_Papíru - Čti_Další_Znak * (432 děl 216);
    Platí_li x <= 0 tak
      OU.Posuň_Papír_Na (0);
    jinak
      OU.Posuň_Papír_Na (x);
    Konec {Platí_li};
  Konec;
06Ch: { ESC 'l' }
  Začátek
    Nastav_Okraje (Omez (Čti_Další_Znak), RVP.Pravý_Okraj);
  Konec;
```

```
  070h: { ESC 'p' }
    Začátek
      GZ.Proporcionální Typ bajt := Čti_Další_Znak mod 2;
    Konec;
  073h: { ESC 's' }
    Začátek
      RVP.Pomalý_Pohyb Typ bajt := Čti_Další_Znak mod 2;
    Konec;
  074h: { ESC 't' }
    Začátek
      GZ.Gen [GZ.ROM, 80h] Typ GZ.Půl_Gen
        := GZTAB.IBM [Low (128 * Čti_Další_Znak)] Typ GZ.Půl_Gen;
    Konec;
  078h: { ESC 'x' }
    Začátek
      GZ.Pečlivý Typ bajt := Čti_Další_Znak mod 2;
    Konec;
  07Eh: { ESC '~' }
    Začátek
      Platí_li (Čti_Další_Znak mod 2) <> 0 tak
        GZ.Gen [GZ.ROM, poř_č ('0')] := GZTAB.IBM [01Fh];
      jinak
        GZ.Gen [GZ.ROM, poř_č ('0')] :=
          GZTAB.IBM [poř_č ('0')];
      Konec {Platí_li};
    Konec;
  v_ostatních_případech { neočekávaný znak }
    Začátek
      RVP.Tiskni_Kus_Řádku;
      RU.Přidej_Do_VP (OU.Nezdar); { ohlas chybu }
    Konec;
  Konec {Vyber};
Konec {Zpracuj_ESC};

{************************************************************}

Začátek {Zpracuj_Znak}
  { !!! hodnota proměnné Hustota_Grafiky !!! }
  { !!! bude předem pevně nastavena      !!! }
  Hustota_Grafiky [0].Skutečná := 24;
  Hustota_Grafiky [0].Požadovaná := 24;
  Hustota_Grafiky [1].Skutečná := 12;
  Hustota_Grafiky [1].Požadovaná := 12;
  Hustota_Grafiky [2].Skutečná := 24;
  Hustota_Grafiky [2].Požadovaná := 12;
  Hustota_Grafiky [3].Skutečná := 12;
  Hustota_Grafiky [3].Požadovaná := 6;
  Hustota_Grafiky [4].Skutečná := 12;
  Hustota_Grafiky [4].Požadovaná := 18;
  Hustota_Grafiky [5].Skutečná := 12;
  Hustota_Grafiky [5].Požadovaná := 20;
  Hustota_Grafiky [6].Skutečná := 12;
  Hustota_Grafiky [6].Požadovaná := 16;
  Hustota_Grafiky [7].Skutečná := 12;
  Hustota_Grafiky [7].Požadovaná := 10;
  GZ.Pečlivý := RP.Nastav;
```

```
1: { programové počáteční nastavení ESC '@' }

  IBM := HW.IBM je_prvkem Předvolba;
  OU.Nastav (HW.Papír_12 je_prvkem Předvolba,
    HW.Přeskok_Perforace je_prvkem Předvolba);
  RVP.Nastav; { ! Hlava musí být na levém kraji ! }
  GZ.Nastav (IBM, HW.KOI je_prvkem Předvolba);

 { Počáteční nastavení proměnných }
  Hustota_KLYZ [0Bh] := 0; { esc K }
  Hustota_KLYZ [0Ch] := 1; { esc L }
  Hustota_KLYZ [09h] := 2; { esc Y }
  Hustota_KLYZ [0Ah] := 3; { esc Z }
  Nastav_Bit_8 := 0;
  Zruš_Bit_8 := 0FFh;
  Auto_LF := nepravda;
  Předvolba_Řádkování := 432 děl 6;
  Řídicí_Taky_Přes_80h := pravda;
  NUL_CAN_Řídicí := pravda;

 { Cyklus na zpracování znaku }
  Odtud_opakuj
    Platí_li RZ.Je_Znak_Ve_VP tak
      RP.Ošetři;
    jinak
      Platí_li Bezprostřední_Tisk tak
        RVP.Tiskni_Kus_Řádku;
        OU.Posuň_Papír_Na (OU.Poloha_Papíru + 216);
        OU.Vykonej_Posuny (0);
      Konec {Platí_li};
      Odtud_opakuj
        RP.Ošetři;
      až_bude_platit RZ.Je_Znak_Ve_VP;
      Platí_li Bezprostřední_Tisk tak
        OU.Posuň_Papír_Na (OU.Poloha_Papíru - 216);
      Konec {Platí_li};
    Konec {Platí_li};

   { Zpracuj znak z rozhraní znaků }
    Čti_Znak;
    Platí_li Řídicí (Přečtený_Znak) tak
      Vyber Přečtený_Znak mod 080h ze
      000h: { NUL }
        Začátek
          { nic nedělej };
        Konec;
      007h: { BEL }
        Začátek
          RVP.Tiskni_Kus_Řádku;
          RU.Přidej_Do_VP (OU.Zvonek);
        Konec;
      008h: { BS }
        Začátek
          RVP.Tiskni_Kus_Řádku; { vytiskni předchozí znaky }
          RVP.Smaž_Znak; { další znak se vytiskne přes }
        Konec;
```

```
009h: { TAB - zpracovává se jako tištitelný znak }
  Začátek
    GZ.Zpracuj_Znak (9);
  Konec;
00Ah: { LF }
  Začátek
    RVP.Tiskni_Kus_Řádku;
    OU.Řádkuj;
  Konec;
00Bh: { VT }
  Začátek
    RVP.Tiskni_Kus_Řádku;
    OU.Tabuluj_Svisle;
  Konec;
00Ch: { FF }
  Začátek
    RVP.Tiskni_Kus_Řádku;
    OU.Stránkuj;
  Konec;
00Dh: { CR }
  Začátek
    RVP.Tiskni_A_Doleva;
    GZ.Široké_Do_Konce_Řádku := nepravda;
    Platí_li Auto_LF tak
      OU.Řádkuj;
    Konec {Platí_li};
  Konec;
00Eh: { SO }
  Začátek
    GZ.Široké_Do_Konce_Řádku := pravda;
  Konec;
00Fh: { SI }
  Začátek
    GZ.Zhuštěný := pravda;
  Konec;
012h: { DC2 }
  Začátek
    GZ.Zhuštěný := nepravda;
  Konec;
014h: { DC4 }
  Začátek
    GZ.Široké_Do_Konce_Řádku := nepravda;
  Konec;
018h: { CAN }
  Začátek
    RVP.Smaž_Všechno;
  Konec;
01Bh: { ESC }
  Začátek
    Zpracuj_ESC;
  Konec;
v_ostatních_případech { neznámý řídicí znak }
  Začátek
    Platí_li NUL_CAN_Řídicí tak
      RVP.Tiskni_Kus_Řádku;
      RU.Přidej_Do_VP (OU.Nezdar);
    jinak
      GZ.Zpracuj_Znak (Přečtený_Znak);
    Konec {Platí_li};
```

```
        Konec;
      Konec {Vyber};
    jinak_platí_li (Přečtený_Znak = 07Fh)
        nebo (Řídicí_Taky_Přes_80h a_zároveň
        (Přečtený_Znak = 0FFh)) tak
      RVP.Smaž_Znak;
    jinak { tištitelný znak }
      GZ.Zpracuj_Znak (Přečtený_Znak);
    Konec {Platí_li};
  až_bude_platit nepravda;
Konec {Zpracuj_Znak};

{*************************************************************}

Konec { ZZ - zpracování znaků }.
```

```
Modul RP;
{ RP - Rozhraní vrchního panelu }
{ Oprava z 02.06.88/14:16 }

Vývoz
  Kontroluj_Papír, Nastav, Ošetři;

Dovoz
  HW, RU, OU;

{*************************************************************}
{*****                    RP - deklarace                 *****}
{*************************************************************}

Konstanta
  Tl_Ruční      = S3; { přepíná provoz ruční / spřažený }
  Tl_Posun      = S4; { ruční posun papíru }
  Světlo_Ruční = S7; { světýlko "RUČNÍ" }
  Došel_Papír   = S2; { snímač konce papíru }

Proměnná
  Kontroluj_Papír: logický; { povolení kontroly papíru }

{*************************************************************}
{*****                    RP - procedury                 *****}
{*************************************************************}

Funkce Nastav: logický;
{ počáteční nastavení - vrací, zda se má tisknout pečlivě }
Začátek
  Kontroluj_Papír := pravda;
  Platí_li Tl_Posun je_prvkem HW.Stav tak
    Nastav := pravda;
    Přidej_Do_VP (OU.Zvonek);
  jinak
    Nastav := nepravda;
  Konec {Platí_li};
Konec {Nastav};

{*************************************************************}

Procedura Ošetři;
{ ošetřuje vrchní panel, předpokládá pravidelné volání }
{ od normy EPSON se liší tím, že nevysílá signál ERR }
Proměnná
  i: celý; { počitadlo sloužící k odměření 1 s }

{=============================================================}

Funkce Chybí_Papír: logický;
Začátek
  Chybí_Papír
    := Kontroluj_Papír a_zároveň
    (Došel_Papír je_prvkem HW.Stav);
Konec {Chybí_Papír};

{*************************************************************}
```

```
Začátek
  Platí_li (Tl_Ruční je_prvkem HW.Stav) nebo (Chybí_Papír) tak
    HW.Řízení_8255 := 2 * poř_č (Světlo_Ruční) + 1; {rozsvit}
    Odtud_opakuj
    až_bude_platit neplatí (Tl_Ruční je_prvkem HW.Stav);
    Platí_li (Chybí_Papír) tak
      Přidej_Do_VP (OU.Došel_Papír);
    Konec {Platí_li};
    Odtud_opakuj
      Platí_li Tl_Posun je_prvkem HW.Stav tak
        i := 15000; { krát 70 us }
        Odtud_opakuj
          i := i - 1;
        až_bude_platit (i = 0) nebo
          neplatí (Tl_Posun je_prvkem HW.Stav);
        { buď vypršel čas (asi 1 s) nebo bylo puštěno tlačítko }
        Platí_li Tl_Posun je_prvkem HW.Stav tak
          OU.Stránkuj;
        jinak
          OU.Řádkuj;
        Konec {Platí_li};
      Konec {Platí_li};
    až_bude_platit (Tl_Ruční je_prvkem HW.Stav) a_zároveň
      neplatí (Chybí_Papír);
    Odtud_opakuj
    až_bude_platit neplatí (Tl_Ruční je_prvkem HW.Stav);
    HW.Řízení_8255 := 2 * poř_č (Světlo_Ruční) + 0; { zhasni }
  Konec {Platí_li};
Konec {Ošetři};

{*********************************************************}

Konec { RP - rozhraní vrchního panelu }.
```

```
Modul GZ;
{ Generátor znaků }
{ Oprava z 06.06.88/12:05 }

Vývoz
  Položka, Čtvrt_Gen, Půl_Gen, Generátor, Druh, Počet_Zarážek,
  Příznak, Široké_Do_Konce_Řádku, Široké_Do_Vypnutí, Élite,
  Proporcionální, Pečlivý, Mezi_Sloupce, Zhuštěný, Nadtržený,
  Podtržený, Příznaky, Prostrkání, Vodorovné_Tabulátory,
  Vybraný, Gen, Česky, Smaž_Tabulátory, Plň_Tabulátory,
  Nastav, Zpracuj_Znak;

Dovoz
  HW, GZTAB, RVP, RU, OU;

Typ
  Položka =
    záznam
      Popis_Tvaru: bajt;
        { = 128 * poř_č (vrchní , ne spodní) osmice jehel }
        { +  16 * první sl. proporc. tisku !je_prvkem [0..7]!}
        { +       posl. sloup. pr. tisku !je_prvkem [4..11]!}
      Tvar_Znaku: pole [0..4] ze Osmice_Jehel;
    Konec {Položka};

  Čtvrt_Gen = pole [0.. 63] ze Položka;

  Půl_Gen   = pole [0..127] ze Položka;

  Generátor = pole [0..255] ze Položka;

  Druh = (ROM, RWM);

Konstanta
  Počet_Zarážek = 40;

Typ
  Příznak =
   (Základní,        { tisk bez proložení linek }
    Mezi_Linky,      { opakování linek o půl linky níž }
    Horní_Index,     { tisk do vrchních 5 linek a mezi ně }
    Dolní_Index);    { tisk do spodních 5 linek a mezi ně }

Proměnná
  Široké_Do_Konce_Řádku, { tiskni každý sloupec dvakrát }
  Široké_Do_Vypnutí,     { tiskni každý sloupec dvakrát }
  Élite,           { vynech 2 sloupce v mezeře mezi znaky }
  Proporcionální, { tiskni znaky v jejich skut. šířce }
  Pečlivý,         { generuj pěkná písmena }
  Mezi_Sloupce,    { generuj sudé sloupce mezi liché }
  Zhuštěný,        { generuj jenom 5 sloupců na znak }
  Nadtržený,              { přidávej nejvrchnější linku }
  Podtržený: logický;     { přidávej nejspodnější linku }
  Příznaky: Příznak;
  Prostrkání: bajt;       { viz ESC mezera }
  Vodorovné_Tabulátory: pole [1..Počet_Zarážek] ze celý;
  Vybraný: Druh; { index právě vybraného generátoru z Gen }
  Gen: pole [Druh] ze Generátor; {_ROM a RWM gen. }

{*********************************************************}
```

```
Procedura Česky;
{ do Gen [ROM] naskládá generátor KOI-8čs }
Proměnná
  i: bajt;
Začátek
  Pro i := 0C1h vzestupně_do 0FEh opakuj
    Gen [ROM, i] := GZTAB.IBM [GZTAB.KOI_8_čs [i]];
  Konec {Pro};
Konec {Česky};

{************************************************************}

Procedura Smaž_Tabulátory;
{ smaže všechny zarážky vodorovného tabulátoru }
Proměnná
  i: bajt;
Začátek
  Pro i := 1 vzestupně_do Počet_Zarážek opakuj
    Vodorovné_Tabulátory [i] := 0;
  Konec {Pro};
Konec {Smaž_Tabulátory};

{************************************************************}

Procedura Plň_Tabulátory;
{ nastaví zarážky vodorovného tabulátoru po 8 pozicích pica }
Proměnná
  i: bajt;
  t: celý;
Začátek
  Smaž_Tabulátory;
  t := 8 * 12;
  Pro i := 1 vzestupně_do 31 { aby t < 255 } opakuj
    Vodorovné_Tabulátory [i] := t;
    t := t + 8 * 12;
  Konec {Pro};
Konec {Plň_Tabulátory};

{************************************************************}

Procedura Zpracuj_Znak (X: bajt);
{ generuje znak X do RVP }
Proměnná
  Uvnitř_Slova: logický;      { pro rozpoznávání hranic slov }
  Posun_Na_Jehlách: logický; { má se posunout o jehlu výš ? }
  První_Sloupec: RVP.Popis_Vršku; {příznaky pro první sloupec}
  Každý_Sloupec: RVP.Popis_Vršku; {příznaky pro každý sloupec}
  Pátý_Sloupec, Šestý_Sloupec: Osmice_Jehel;
  Položka: RVP.Položka;
  i, j, l, p: bajt;
  t: celý;

{============================================================}
```

```
Funkce Levý_Sloupec (Kód_Znaku: bajt): bajt;
{ vrátí číslo levého krajního sloupce znaku, }
{ které je_prvkem [0..3]          }

Začátek
  Platí_li Proporcionální tak
    Levý_Sloupec := Gen [Vybraný, Kód_Znaku].Popis_Tvaru
                    mod 128 děl 32;
  jinak
    Levý_Sloupec := 0;
  Konec {Platí_li};
Konec {Levý_Sloupec};

{************************************************************}

Funkce Pravý_Sloupec (Kód_Znaku: bajt): bajt;
{ vrátí číslo levého krajního sloupce znaku, }
{ které je_prvkem [0..5]          }
Proměnná
  X: bajt;
Začátek
  X := Gen [Vybraný, Kód_Znaku].Popis_Tvaru děl 2 mod 8;
  Platí_li X >= 5 tak
    Pravý_Sloupec := 5; { 5. sloupec opakuje 3. sloupec }
  jinak_platí_li Proporcionální tak
    Pravý_Sloupec := X;
  jinak
    Pravý_Sloupec := 4;
  Konec { Platí_li };
Konec {Pravý_Sloupec};

{************************************************************}

Procedura Podtrhni;
Začátek
  Platí_li Nadtržený tak
    Položka [RVP.Základ].Vršek :=
      Položka [RVP.Základ].Vršek + [RVP.Jehla_9];
  Konec {Platí_li};
  Platí_li Podtržený tak
    Položka [RVP.Základ].Spodek :=
      Položka [RVP.Základ].Spodek + [J1];
  Konec {Platí_li};
  RVP.Přidej_Sloupec (Položka);
Konec {Podtrhni};

{************************************************************}

Procedura Prázdný_Sloupec (P: RVP.Popis_Vršku);
Začátek
  Položka [RVP.Základ].Spodek := [];
  Položka [RVP.Základ].Vršek := P;
  Položka [RVP.Přetisk].Spodek := [];
  Položka [RVP.Přetisk].Vršek := [];
  Podtrhni;
Konec {Prázdný_Sloupec};

{************************************************************}
```

```
Procedura Generuj_Sloupce
  (LS {levý sloupec}, PS {pravý sloupec}: Osmice_Jehel;
   P: RVP.Popis_Vršku);
{ z 1 sloupce generátoru LS vygeneruje potřebné položky RVP }
Proměnná
  Posunutý_Levý, Posunutý_Pravý: Osmice_Jehel;

{=========================================================}

Procedura Jednoduché_Sloupce (P: RVP.Popis_Vršku);
{ generuje sloupce nerozšířeného písma }

{=========================================================}

Procedura Generuj_Linky (Základ, Přetisk: Osmice_Jehel);
{ generuje potřebné linky do jednoho sloupce }

{=========================================================}

Procedura Indexuj;
Proměnná
  Sloupec_Z_Generátoru: Osmice_Jehel;
Začátek
  Sloupec_Z_Generátoru := Položka [RVP.Základ].Spodek;
  Položka [RVP.Základ].Spodek := [];
  Položka [RVP.Základ].Vršek := [];
  Položka [RVP.Přetisk].Spodek := [];
  Položka [RVP.Přetisk].Vršek := [];
  Platí_li J1 je_prvkem Sloupec_Z_Generátoru tak
    Položka [RVP.Přetisk].Spodek :=
      Položka [RVP.Přetisk].Spodek + [J1];
  Konec {Platí_li};
  Platí_li J2 je_prvkem Sloupec_Z_Generátoru tak
    Položka [RVP.Základ].Spodek :=
      Položka [RVP.Základ].Spodek + [J1];
  Konec {Platí_li};
  Platí_li J3 je_prvkem Sloupec_Z_Generátoru tak
    Položka [RVP.Přetisk].Spodek :=
      Položka [RVP.Přetisk].Spodek + [J2];
  Konec {Platí_li};
  Platí_li J4 je_prvkem Sloupec_Z_Generátoru tak
    Položka [RVP.Základ].Spodek :=
      Položka [RVP.Základ].Spodek + [J2];
  Konec {Platí_li};
  Platí_li J5 je_prvkem Sloupec_Z_Generátoru tak
    Položka [RVP.Přetisk].Spodek :=
      Položka [RVP.Přetisk].Spodek + [J3];
  Konec {Platí_li};
  Platí_li J6 je_prvkem Sloupec_Z_Generátoru tak
    Položka [RVP.Základ].Spodek :=
      Položka [RVP.Základ].Spodek + [J3];
  Konec {Platí_li};
  Platí_li J7 je_prvkem Sloupec_Z_Generátoru tak
    Položka [RVP.Přetisk].Spodek :=
      Položka [RVP.Přetisk].Spodek + [J4];
  Konec {Platí_li};
```

```
  Platí_li J8 je_prvkem Sloupec_Z_Generátoru tak
    Položka [RVP.Základ].Spodek :=
      Položka [RVP.Základ].Spodek + [J4];
  Konec {Platí_li};

  { pozn.: přetypování na bajt nebo celé a násobení dvěma  }
  { způsobí posun černých teček o 1 jehlu vzhůru, násobení }
  { šestnácti o 4 a násobení 32 o 5 jehel vzhůru           }
  Platí_li Posun_Na_Jehlách tak
    Platí_li Příznaky = Dolní_Index tak
      Položka [RVP.Přetisk].Spodek Typ celé :=
        Položka [RVP.Přetisk].Spodek Typ celé * 2;
    jinak
      Položka [RVP.Základ].Spodek Typ celé :=
        Položka [RVP.Základ].Spodek Typ celé * 32;
      Položka [RVP.Přetisk].Spodek Typ celé :=
        Položka [RVP.Přetisk].Spodek Typ celé * 32;
    Konec {Platí_li};
  jinak
    Platí_li Příznaky = Dolní_Index tak
      { je to v pořádku, nic nedělej }
    jinak
      Položka [RVP.Základ].Spodek Typ celé :=
        Položka [RVP.Základ].Spodek Typ celé * 16;
      Položka [RVP.Přetisk].Spodek Typ celé :=
        Položka [RVP.Přetisk].Spodek Typ celé * 32;
    Konec {Platí_li};
  Konec {Platí_li};
Konec {Indexuj};

{************************************************************}

Začátek {Generuj_Linky}
  Položka [RVP.Základ].Spodek := Základ;
  Položka [RVP.Základ].Vršek := [];
  Položka [RVP.Přetisk].Vršek := [];
  Platí_li Příznaky >= Horní_Index tak
    Indexuj;
  jinak
    Položka [RVP.Přetisk].Spodek := Přetisk;
    Platí_li Posun_Na_Jehlách tak
      Položka [RVP.Základ].Spodek Typ celý :=
        Položka [RVP.Základ].Spodek Typ celý * 2;
      Položka [RVP.Přetisk].Spodek Typ celý :=
        Položka [RVP.Přetisk].Spodek Typ celý * 2;
    Konec {Platí_li};
  Konec {Platí_li};
  Položka [RVP.Základ].Vršek :=
    Položka [RVP.Základ].Vršek + P;
  Podtrhni;
Konec {Generuj_Linky};

{************************************************************}

Začátek {Jednoduché_Sloupce}
  Posunutý_Levý Typ bajt := 2 * LS Typ bajt;
  Posunutý_Pravý Typ bajt := 2 * PS Typ bajt;
```

```
  Platí_li Příznaky = Mezi_Linky tak
    Generuj_Linky (LS, LS);
  jinak_platí_li Pečlivý tak
    Generuj_Linky (LS, Posunutý_Levý * LS);
  jinak
    Generuj_Linky (LS, []);
  Konec {Platí_li};
  Platí_li neplatí Zhuštěný tak
    P := Každý_Sloupec;
    Platí_li GZ.Mezi_Sloupce tak
    { má přednost před pečlivým tiskem }
      Platí_li Příznaky = Mezi_Linky tak
        Generuj_Linky (LS, LS);
      jinak
        Generuj_Linky (LS, []);
      Konec {Platí_li};
    jinak_platí_li Pečlivý tak
      Generuj_Linky
        (LS * PS, Posunutý_Levý * PS + LS *  Posunutý_Pravý);
    jinak
      Generuj_Linky ([], []);
    Konec {Platí_li};
  Konec {Platí_li};
Konec {Jednoduché_Sloupce};

{***********************************************************}

Začátek {Generuj_Sloupce}
  Jednoduché_Sloupce (P);
  Platí_li Široké_Do_Konce_Řádku nebo Široké_Do_Vypnutí tak
    Jednoduché_Sloupce (Každý_Sloupec);
  Konec {Platí_li};
Konec {Generuj_Sloupce};

{***********************************************************}

Začátek {Zpracuj_Znak}
  Platí_li RVP.První_Prázdná = RVP.První_Plná tak
    Uvnitř_Slova := nepravda;
  Konec {Platí_li};
  Platí_li X = 9 { tabulátor } tak
    { najdi nejbližší zarážku vodorovného tabulátoru }
    j := 0;
    Odtud_opakuj
      j := j + 1;
      Platí_li j > Počet_Zarážek tak
        t := 0;
      jinak
        t := Vodorovné_Tabulátory [j];
      Konec {Platí_li};
    až_bude_platit (t > RVP.První_Prázdná) nebo (t = 0);
    { naplň RVP prázdnými sloupci až k poloze zarážky }
    Platí_li Uvnitř_Slova tak
      První_Sloupec := [RVP.Začátek.Mezer,
                        RVP.Hranice_Znaku,
                        RVP.Tabulátor];
      Uvnitř_Slova := nepravda;
    jinak
      První_Sloupec := [RVP.Hranice_Znaku, RVP.Tabulátor];
    Konec {Platí_li};
```

```
    t := t - RVP.První_Prázdná;
    Platí_li t > 0 tak
      Prázdný_Sloupec (První_Sloupec);
      Platí_li t > 1 tak
        Pro j := 2 vzestupně_do t opakuj
          Prázdný_Sloupec ([]);
        Konec {Pro};
      Konec {Platí_li};
    Konec {Platí_li};
  jinak { není to tabulátor }
    Posun_Na_Jehlách Typ bajt :=
      Gen [Vybraný, X].Popis_Tvaru děl 128;
    p := Pravý_Sloupec (X);
    l := Levý_Sloupec (X);
    Platí_li X = poř_č (' ') tak
      Každý_Sloupec := [];
      Platí_li Uvnitř_Slova tak
        První_Sloupec := [RVP.Hranice_Znaku,
                          RVP.Mezera,
                          RVP.Začátek.Mezer];
        Uvnitř_Slova := nepravda;
      jinak
        První_Sloupec := [RVP.Hranice_Znaku, RVP.Mezera];
      Konec {Platí_li};
    jinak { není to mezera }
      Platí_li RVP.První_Prázdná > RVP.Pravý_Okraj tak
        RVP.Vyprázdni_Přeplněnou;
        OU.Řádkuj;
      Konec {Platí_li};
      Platí_li Uvnitř_Slova tak
        První_Sloupec := [RVP.Hranice_Znaku];
      jinak
        První_Sloupec := [RVP.Hranice_Znaku, RVP.Začátek_Slova];
        Uvnitř_Slova := pravda;
      Konec {Platí_li};
      Platí_li p = 5 tak
       { je to grafický znak na 12 linek }
        Každý_Sloupec := [RVP.Dvanáct_Linek];
        p := 4;
      jinak
        Každý_Sloupec := [];
      Konec {Platí_li};
    Konec {Platí_li};
    Generuj_Sloupce
     (Gen [Vybraný, X].Tvar_Znaku [l],
      Gen [Vybraný, X].Tvar_Znaku [l+1],
      Každý_Sloupec + První_Sloupec);
    Pro i := l + 1 vzestupně_do p - 1 opakuj
      Generuj_Sloupce
       (Gen [Vybraný, X].Tvar_Znaku [i],
        Gen [Vybraný, X].Tvar_Znaku [i+1],
        Každý_Sloupec);
    Konec {Pro};
    Platí_li Pravý_Sloupec (X) < 5 tak
      { negraf. znaky prostrč prázdným místem }
      Pátý_Sloupec := [];
      Šestý_Sloupec := [];
```

```
    jinak
      { graf. znaky prostrč 3. a 4. sloupcem }
      Pátý_Sloupec := Gen [Vybraný, X].Tvar_Znaku [3];
      Šestý_Sloupec := Gen [Vybraný, X].Tvar_Znaku [4];
    Konec {Platí_li};
    Generuj_Sloupce (Gen [Vybraný, X].Tvar_Znaku [p],
      Pátý_Sloupec, Každý_Sloupec);
    Platí_li Zhuštěný nebo neplatí Élite tak
      Generuj_Sloupce
        (Pátý_Sloupec, Šestý_Sloupec, Každý_Sloupec);
      Platí_li Zhuštěný a_zároveň neplatí Élite tak
        Generuj_Sloupce (Šestý_Sloupec, [], Každý_Sloupec);
      Konec {Platí_li};
    Konec {Platí_li};
    Pro i := 1 vzestupně_do Prostrkání opakuj
      Prázdný_Sloupec ([]);
    Konec {Pro};
  Konec {Platí_li};
Konec {Zpracuj_Znak};

{************************************************************}

Procedura Nastav (IBM, KOI: logický);
{ počáteční nastavení proměnných modulu }
Proměnná
  i: bajt;
  t: celý;
Začátek
  Široké_Do_Konce_Řádku := nepravda;
  Široké_Do_Vypnutí := nepravda;
  Élite := nepravda;
  Proporcionální := nepravda;
  GZ.Mezi_Sloupce := nepravda;
  Zhuštěný := nepravda;
  Podtržený := nepravda;
  Nadtržený := nepravda;
  Příznaky := Základní;
  Prostrkání := 0;
  Plň_Tabulátory;
  Vybraný := ROM;
  Gen [ROM] := GZTAB.IBM;
  Platí_li neplatí IBM tak
    Gen [ROM, 080h] Typ Půl_Gen := GZ_EPS;
  Konec {Platí_li};
  Platí_li KOI tak
    Česky;
  Konec {Platí_li};
Konec {Nastav};

{************************************************************}

Konec { GZ - generátor znaků }.
```

```
        title 'GZTAB'
; generátor znaků
; Oprava z 01.06.88/14:10
        public IBM, KOI_8_čs, Přeskupení, Národní_Sada
        cseg
IBM:
        db 088h, 01Ch, 0A2h, 062h, 0A2h, 022h
        db 088h, 01Ch, 0A2h, 062h, 0A2h, 0FEh
        db 088h, 01Ch, 0AAh, 06Ah, 0AAh, 01Ah
        db 088h, 03Eh, 010h, 060h, 0A0h, 020h
        db 088h, 03Ch, 002h, 082h, 002h, 03Eh
        db 0A8h, 000h, 082h, 0FEh, 002h, 0C0h
        db 088h, 082h, 0FEh, 042h, 020h, 040h
        db 088h, 03Eh, 0A0h, 060h, 0A0h, 01Eh
        db 088h, 03Eh, 090h, 060h, 0A0h, 020h
        db 088h, 012h, 0AAh, 06Ah, 0AAh, 024h
        db 088h, 020h, 020h, 0FCh, 022h, 0C2h
        db 088h, 039h, 005h, 042h, 084h, 038h
        db 088h, 022h, 0A6h, 06Ah, 0B2h, 022h
        db 088h, 000h, 080h, 040h, 080h, 000h
        db 088h, 0BEh, 048h, 088h, 048h, 03Eh
        db 088h, 03Eh, 048h, 088h, 048h, 0BEh
; 010h:
        db 088h, 07Ch, 082h, 042h, 082h, 044h
        db 088h, 0FEh, 082h, 042h, 084h, 038h
        db 088h, 0FEh, 092h, 052h, 082h, 082h
        db 088h, 0FEh, 090h, 098h, 054h, 0A2h
        db 088h, 082h, 0FEh, 082h, 000h, 0C0h
        db 088h, 052h, 0AAh, 0AAh, 0AAh, 094h
        db 088h, 0FCh, 002h, 082h, 002h, 0FCh
        db 088h, 0FEh, 002h, 002h, 002h, 0C2h
        db 088h, 0FEh, 002h, 082h, 042h, 082h
        db 088h, 0FEh, 020h, 090h, 048h, 0BEh
        db 088h, 07Ch, 082h, 082h, 042h, 0BCh
        db 088h, 01Ch, 062h, 0A2h, 062h, 01Ch
        db 088h, 062h, 092h, 052h, 092h, 08Ch
        db 088h, 080h, 080h, 0FEh, 080h, 0E0h
        db 088h, 0FCh, 002h, 0E2h, 002h, 0FCh
        db 088h, 03Ah, 04Ch, 092h, 064h, 0B8h
; 020h:
        db 088h, 000h, 000h, 000h, 000h, 000h
        db 0A6h, 000h, 000h, 0FAh, 000h, 000h
        db 088h, 000h, 0C0h, 000h, 0C0h, 000h
        db 088h, 028h, 07Ch, 028h, 07Ch, 028h
        db 088h, 024h, 054h, 0FEh, 054h, 048h
        db 088h, 0C6h, 0C8h, 010h, 026h, 0C6h
        db 088h, 06Ch, 092h, 09Ah, 064h, 00Ah
        db 025h, 000h, 0D0h, 0E0h, 000h, 000h
        db 026h, 000h, 038h, 044h, 082h, 000h
        db 026h, 000h, 082h, 044h, 038h, 000h
        db 088h, 028h, 010h, 07Ch, 010h, 028h
        db 088h, 010h, 010h, 07Ch, 010h, 010h
        db 0A4h, 000h, 00Dh, 00Eh, 000h, 000h
        db 088h, 010h, 010h, 010h, 010h, 010h
        db 0A4h, 000h, 006h, 006h, 000h, 000h
        db 088h, 006h, 008h, 010h, 020h, 0C0h
; 030h:
        db 088h, 038h, 044h, 082h, 044h, 038h
        db 086h, 010h, 022h, 0FEh, 002h, 000h
```

```
        db 088h, 046h, 08Ah, 092h, 092h, 062h
        db 088h, 084h, 082h, 0A2h, 0E2h, 09Ch
        db 088h, 018h, 068h, 088h, 0FEh, 008h
        db 088h, 0E4h, 0A2h, 0A2h, 0A2h, 09Ch
        db 088h, 03Ch, 052h, 092h, 092h, 08Ch
        db 088h, 086h, 088h, 090h, 0A0h, 0C0h
        db 088h, 06Ch, 092h, 092h, 092h, 06Ch
        db 088h, 062h, 092h, 092h, 094h, 078h
        db 0A4h, 000h, 066h, 066h, 000h, 000h
        db 0A4h, 000h, 06Dh, 06Eh, 000h, 000h
        db 086h, 010h, 028h, 044h, 082h, 000h
        db 088h, 028h, 028h, 028h, 028h, 028h
        db 0A8h, 000h, 082h, 044h, 028h, 010h
        db 088h, 040h, 080h, 08Ah, 090h, 060h
; 040h:
        db 088h, 09Eh, 092h, 09Eh, 082h, 07Ch
        db 088h, 03Eh, 048h, 088h, 048h, 03Eh
        db 088h, 0FEh, 092h, 092h, 092h, 06Ch
        db 088h, 07Ch, 082h, 082h, 082h, 044h
        db 088h, 0FEh, 082h, 082h, 044h, 038h
        db 088h, 0FEh, 092h, 092h, 082h, 082h
        db 088h, 0FEh, 090h, 090h, 080h, 080h
        db 088h, 07Ch, 082h, 092h, 092h, 09Ch
        db 088h, 0FEh, 010h, 010h, 010h, 0FEh
        db 0A6h, 000h, 082h, 0FEh, 082h, 000h
        db 088h, 004h, 002h, 002h, 002h, 0FCh
        db 088h, 0FEh, 010h, 028h, 044h, 082h
        db 088h, 0FEh, 002h, 002h, 002h, 002h
        db 088h, 0FEh, 040h, 030h, 040h, 0FEh
        db 088h, 0FEh, 020h, 010h, 008h, 0FEh
        db 088h, 07Ch, 082h, 082h, 082h, 07Ch
; 050h:
        db 088h, 0FEh, 090h, 090h, 090h, 060h
        db 088h, 07Ch, 082h, 08Ah, 084h, 07Ah
        db 088h, 0FEh, 090h, 098h, 094h, 062h
        db 088h, 062h, 092h, 092h, 092h, 08Ch
        db 088h, 080h, 080h, 0FEh, 080h, 080h
        db 088h, 0FCh, 002h, 002h, 002h, 0FCh
        db 088h, 0F0h, 008h, 006h, 008h, 0F0h
        db 088h, 0FCh, 002h, 01Ch, 002h, 0FCh
        db 088h, 0C6h, 028h, 010h, 028h, 0C6h
        db 088h, 0C0h, 020h, 01Eh, 020h, 0C0h
        db 088h, 086h, 08Ah, 092h, 0A2h, 0C2h
        db 0A6h, 000h, 0FEh, 082h, 082h, 000h
        db 088h, 0C0h, 020h, 010h, 008h, 006h
        db 0A6h, 000h, 082h, 082h, 0FEh, 000h
        db 088h, 020h, 040h, 080h, 040h, 020h
        db 008h, 001h, 001h, 001h, 001h, 001h
; 060h:
        db 0C6h, 000h, 000h, 0E0h, 0D0h, 000h
        db 088h, 004h, 02Ah, 02Ah, 02Ah, 01Eh
        db 088h, 0FEh, 022h, 022h, 022h, 01Ch
        db 088h, 01Ch, 022h, 022h, 022h, 022h
        db 088h, 01Ch, 022h, 022h, 022h, 0FEh
        db 088h, 01Ch, 02Ah, 02Ah, 02Ah, 01Ah
        db 086h, 010h, 07Eh, 090h, 090h, 000h
        db 008h, 038h, 045h, 045h, 045h, 07Eh
        db 088h, 0FEh, 020h, 020h, 020h, 01Eh
        db 0A6h, 000h, 022h, 0BEh, 002h, 000h
```

```
        db 086h, 001h, 001h, 001h, 0BEh, 000h
        db 088h, 0FEh, 008h, 008h, 014h, 022h
        db 0A6h, 000h, 082h, 0FEh, 002h, 000h
        db 088h, 03Eh, 020h, 01Eh, 020h, 01Eh
        db 088h, 03Eh, 020h, 020h, 020h, 01Eh
        db 088h, 01Ch, 022h, 022h, 022h, 01Ch
; 070h:
        db 008h, 07Fh, 044h, 044h, 044h, 038h
        db 008h, 038h, 044h, 044h, 044h, 07Fh
        db 088h, 03Eh, 010h, 020h, 020h, 020h
        db 088h, 012h, 02Ah, 02Ah, 02Ah, 024h
        db 088h, 020h, 020h, 0FCh, 022h, 022h
        db 088h, 03Ch, 002h, 002h, 002h, 03Eh
        db 088h, 030h, 008h, 006h, 008h, 030h
        db 088h, 03Ch, 002h, 00Ch, 002h, 03Ch
        db 088h, 022h, 014h, 008h, 014h, 022h
        db 008h, 071h, 00Ah, 004h, 008h, 070h
        db 088h, 022h, 026h, 02Ah, 032h, 022h
        db 088h, 010h, 010h, 06Ch, 082h, 082h
        db 0A6h, 000h, 000h, 0EEh, 000h, 000h
        db 088h, 082h, 082h, 06Ch, 010h, 010h
        db 088h, 060h, 080h, 040h, 020h, 0C0h
        db 088h, 0E0h, 010h, 0CEh, 010h, 0E0h
; 080h:
        db 008h, 078h, 085h, 085h, 086h, 048h
        db 088h, 03Ch, 082h, 002h, 082h, 03Eh
        db 088h, 01Ch, 02Ah, 06Ah, 0AAh, 01Ah
        db 088h, 004h, 06Ah, 0AAh, 06Ah, 01Eh
        db 088h, 004h, 0AAh, 02Ah, 0AAh, 01Eh
        db 088h, 004h, 0AAh, 06Ah, 02Ah, 01Eh
        db 088h, 004h, 02Ah, 0AAh, 02Ah, 01Eh
        db 008h, 038h, 045h, 045h, 046h, 044h
        db 088h, 01Ch, 06Ah, 0AAh, 06Ah, 01Ah
        db 088h, 01Ch, 0AAh, 02Ah, 0AAh, 01Ah
        db 088h, 01Ch, 0AAh, 06Ah, 02Ah, 01Ah
        db 0A6h, 000h, 0A2h, 03Eh, 082h, 000h
        db 0A6h, 000h, 062h, 0BEh, 042h, 000h
        db 0A6h, 000h, 0A2h, 07Eh, 002h, 000h
        db 088h, 0BEh, 048h, 088h, 048h, 0BEh
        db 088h, 00Eh, 054h, 0A4h, 054h, 00Eh
; 090h:
        db 088h, 0FEh, 092h, 012h, 042h, 082h
        db 088h, 024h, 02Ah, 01Eh, 02Ah, 01Ah
        db 088h, 07Eh, 090h, 0FEh, 092h, 082h
        db 088h, 00Ch, 052h, 092h, 052h, 00Ch
        db 088h, 01Ch, 0A2h, 022h, 0A2h, 01Ch
        db 088h, 01Ch, 0A2h, 062h, 022h, 01Ch
        db 088h, 03Ch, 042h, 082h, 042h, 03Eh
        db 088h, 03Ch, 082h, 042h, 002h, 03Eh
        db 088h, 038h, 085h, 002h, 084h, 038h
        db 088h, 0BCh, 042h, 082h, 042h, 0BCh
        db 088h, 07Ch, 0C2h, 002h, 0C2h, 07Ch
        db 088h, 038h, 044h, 0FEh, 044h, 044h
        db 088h, 012h, 07Eh, 092h, 082h, 042h
        db 088h, 094h, 054h, 03Eh, 054h, 094h
        db 088h, 0FEh, 0A0h, 048h, 01Eh, 008h
        db 088h, 044h, 038h, 028h, 038h, 044h
; 0A0h:
        db 088h, 004h, 02Ah, 06Ah, 0AAh, 01Eh
```

```
        db 0A8h, 000h, 022h, 03Eh, 042h, 080h
        db 088h, 01Ch, 022h, 062h, 0A2h, 01Ch
        db 088h, 03Ch, 002h, 042h, 082h, 03Eh
        db 088h, 0DEh, 090h, 050h, 050h, 08Eh
        db 088h, 0BEh, 090h, 048h, 044h, 0BEh
        db 088h, 012h, 0AAh, 0AAh, 0AAh, 07Ah
        db 088h, 072h, 08Ah, 08Ah, 08Ah, 072h
        db 088h, 00Ch, 012h, 0A2h, 002h, 004h
        db 088h, 038h, 020h, 020h, 020h, 020h
        db 088h, 020h, 020h, 020h, 020h, 038h
        db 088h, 0C8h, 010h, 032h, 056h, 08Ah
        db 088h, 0C8h, 010h, 02Ch, 054h, 0BEh
        db 0A6h, 000h, 000h, 0BEh, 000h, 000h
        db 088h, 010h, 028h, 044h, 092h, 0AAh
        db 088h, 0AAh, 092h, 044h, 028h, 010h
; 0B0h:
        db 08Ah, 0AAh, 000h, 0AAh, 000h, 0AAh
        db 08Ah, 0AAh, 055h, 0AAh, 055h, 0AAh
        db 08Ah, 055h, 0AAh, 055h, 0AAh, 055h
        db 08Ah, 000h, 000h, 0FFh, 000h, 000h
        db 08Ah, 008h, 008h, 0FFh, 000h, 000h
        db 08Ah, 008h, 008h, 0FFh, 000h, 000h
        db 08Ah, 008h, 008h, 0FFh, 000h, 000h
        db 08Ah, 008h, 008h, 00Fh, 000h, 000h
        db 08Ah, 008h, 008h, 00Fh, 000h, 000h
        db 08Ah, 008h, 008h, 0FFh, 000h, 000h
        db 08Ah, 000h, 000h, 0FFh, 000h, 000h
        db 08Ah, 008h, 008h, 00Fh, 000h, 000h
        db 08Ah, 008h, 008h, 0F8h, 000h, 000h
        db 08Ah, 008h, 008h, 0F8h, 000h, 000h
        db 08Ah, 008h, 008h, 0F8h, 000h, 000h
        db 08Ah, 008h, 008h, 00Fh, 000h, 000h
; 0C0h:
        db 08Ah, 000h, 000h, 0F8h, 008h, 008h
        db 08Ah, 008h, 008h, 0F8h, 008h, 008h
        db 08Ah, 008h, 008h, 00Fh, 008h, 008h
        db 08Ah, 000h, 000h, 0FFh, 008h, 008h
        db 08Ah, 008h, 008h, 008h, 008h, 008h
        db 08Ah, 008h, 008h, 0FFh, 008h, 008h
        db 08Ah, 000h, 000h, 0FFh, 008h, 008h
        db 08Ah, 000h, 000h, 0FFh, 008h, 008h
        db 08Ah, 000h, 000h, 0F8h, 008h, 008h
        db 08Ah, 000h, 000h, 00Fh, 008h, 008h
        db 08Ah, 008h, 008h, 0F8h, 008h, 008h
        db 08Ah, 008h, 008h, 00Fh, 008h, 008h
        db 08Ah, 000h, 000h, 0FFh, 008h, 008h
        db 08Ah, 008h, 008h, 008h, 008h, 008h
        db 08Ah, 008h, 008h, 0FFh, 008h, 008h
        db 08Ah, 008h, 008h, 0F8h, 008h, 008h
; 0D0h:
        db 08Ah, 008h, 008h, 0F8h, 008h, 008h
        db 08Ah, 008h, 008h, 00Fh, 008h, 008h
        db 08Ah, 008h, 008h, 00Fh, 008h, 008h
        db 08Ah, 000h, 000h, 0F8h, 008h, 008h
        db 08Ah, 000h, 000h, 0F8h, 008h, 008h
        db 08Ah, 000h, 000h, 00Fh, 008h, 008h
        db 08Ah, 000h, 000h, 00Fh, 008h, 008h
        db 08Ah, 008h, 008h, 0FFh, 008h, 008h
        db 08Ah, 008h, 008h, 0FFh, 008h, 008h
```

```
        db 08Ah, 008h, 008h, 0F8h, 000h, 000h
        db 08Ah, 000h, 000h, 00Fh, 008h, 008h
        db 08Ah, 0FFh, 0FFh, 0FFh, 0FFh, 0FFh
        db 08Ah, 00Fh, 00Fh, 00Fh, 00Fh, 00Fh
        db 08Ah, 0FFh, 0FFh, 0FFh, 000h, 000h
        db 08Ah, 000h, 000h, 0FFh, 0FFh, 0FFh
        db 08Ah, 0F0h, 0F0h, 0F0h, 0F0h, 0F0h
; 0E0h:
        db 088h, 01Ch, 022h, 022h, 01Ch, 022h
        db 088h, 07Fh, 090h, 092h, 072h, 00Ch
        db 088h, 0FEh, 080h, 080h, 0C0h, 000h
        db 088h, 022h, 03Ch, 020h, 03Ch, 022h
        db 088h, 0C6h, 0AAh, 092h, 082h, 082h
        db 088h, 01Ch, 022h, 022h, 03Ch, 020h
        db 008h, 003h, 07Ch, 004h, 004h, 078h
        db 088h, 010h, 020h, 03Fh, 020h, 020h
        db 088h, 092h, 0AAh, 0EEh, 0AAh, 092h
        db 088h, 07Ch, 092h, 092h, 092h, 07Ch
        db 088h, 072h, 08Eh, 080h, 08Eh, 072h
        db 088h, 01Ch, 022h, 062h, 09Ch, 040h
        db 088h, 018h, 024h, 018h, 024h, 018h
        db 008h, 03Ah, 04Ch, 054h, 064h, 0B8h
        db 088h, 038h, 054h, 092h, 092h, 092h
        db 088h, 03Ch, 040h, 040h, 040h, 03Ch
; 0F0h:
        db 088h, 054h, 054h, 054h, 054h, 054h
        db 088h, 024h, 024h, 074h, 024h, 024h
        db 088h, 054h, 054h, 024h, 024h, 024h
        db 088h, 024h, 024h, 024h, 054h, 054h
        db 088h, 000h, 000h, 07Fh, 080h, 040h
        db 088h, 002h, 001h, 0FEh, 000h, 000h
        db 088h, 010h, 010h, 054h, 010h, 010h
        db 088h, 06Ch, 048h, 024h, 024h, 048h
        db 086h, 040h, 0A0h, 0A0h, 040h, 000h
        db 0A4h, 000h, 018h, 018h, 000h, 000h
        db 0A4h, 000h, 008h, 008h, 000h, 000h
        db 088h, 008h, 008h, 004h, 0FEh, 080h
        db 086h, 0F0h, 080h, 080h, 070h, 000h
        db 084h, 090h, 0B0h, 050h, 000h, 000h
        db 0A6h, 000h, 038h, 038h, 038h, 000h
        db 088h, 086h, 08Ah, 052h, 0A2h, 0C2h
```

```
;
; překódování pro KOI-8čs na pozice 0C1h až 0FEh
KOI_8_čs:
;  pole [0C1h..0FEh] ze bajt; { index do IBM }
;
        db      0A0h, 0C2h, 000h, 001h, 002h, 003h, 0C7h
        db 081h, 0A1h, 004h, 005h, 006h, 094h, 007h, 0A2h
        db 093h, 084h, 008h, 009h, 00Ah, 0A3h, 0D6h, 082h
        db 085h, 00Bh, 00Ch, 0DBh, 00Dh, 022h, 05Eh, 0AAh
        db 027h, 00Fh, 0E2h, 010h, 011h, 012h, 013h, 0E7h
        db 09Ah, 014h, 016h, 017h, 018h, 099h, 019h, 01Ah
        db 01Bh, 08Eh, 013h, 01Ch, 01Dh, 01Eh, 015h, 090h
        db 00Eh, 07Fh, 0FFh, 0FBh, 027h, 060h, 0F8h
;
; přeskupení pro národní sady znaků
Přeskupení:
; pole [1..46] ze záznam Odkud, Kam: bajt; Konec;
;
        db 085h, 040h, 087h, 05Ch, 015h, 05Dh, 082h, 07Bh
        db 097h, 07Ch, 00Eh, 07Eh, 0F8h, 05Bh, 08Ah, 07Dh
        db 082h, 05Dh, 097h, 060h, 085h, 07Bh, 095h, 07Ch
        db 08Dh, 07Eh, 015h, 040h, 09Ah, 05Dh, 081h, 07Dh
        db 0E1h, 07Eh, 08Eh, 05Bh, 099h, 05Ch, 084h, 07Bh
        db 094h, 07Ch, 09Fh, 024h, 090h, 040h, 09Ah, 05Eh
        db 082h, 060h, 081h, 07Eh, 08Fh, 05Dh, 086h, 07Dh
        db 092h, 05Bh, 01Fh, 05Ch, 091h, 07Bh, 0EDh, 07Ch
        db 09Eh, 023h, 0ADh, 05Bh, 0A5h, 05Ch, 0A8h, 05Dh
        db 022h, 07Bh, 0A4h, 07Ch, 0A0h, 040h, 082h, 05Eh
        db 0A1h, 07Bh, 0A2h, 07Dh, 0A3h, 07Eh, 081h, 060h
        db 09Ch, 023h, 09Dh, 05Ch
;
; počáteční a koncové indexy do GZ_PRE pro národní sady znaků
Národní_Sada: ; pole [Francie..Lat_Amerika] ze
;                 záznam První, Poslední: bajt; Konec;
        db 01d, 08d ; Francie
        db 14d, 21d ; Německo
        db 45d, 45d ; Británie
        db 27d, 32d ; Dánsko I
        db 18d, 28d ; Švédsko
        db 07d, 13d ; Itálie
        db 33d, 38d ; Španělsko I
        db 46d, 46d ; Japonsko
        db 22d, 32d ; Norsko
        db 23d, 32d ; Dánsko II
        db 34d, 43d ; Španělsko II
        db 34d, 44d ; Latinská Amerika
        end
```

```
Modul RVP;
{ Rádková vyrovnávací paměť }
{ Oprava z 06.06.88/12:07 }

Vývoz
  Pravý_Doraz, Pomalý_Pohyb, Zleva_Doprava, První_Plná,
  První_Prázdná, Levý_Okraj, Pravý_Okraj, Zarovnání, Popis,
  Index_VP, Popis_Vršku, Půlpoložka, Položka, Smaž_Všechno,
  Smaž_Znak, Nastav, Přidej_Sloupec, Hlavu_Doleva,
  Vyprázdni_Přeplněnou, Tiskni_Kus_Řádku, Tiskni_A_Doleva;

Dovoz
  HW, RVP, RU, OU;

Konstanta
  Pravý_Doraz = 959 {kroků}; { mezní počet kroků na řádku }

Typ

  Popis =
   (Jehla_9,          { bouchni 9. jehlou }
    Nic,              { nepoužito }
    Dvanáct_Linek,    {tisk graf. znaku též 4 linky pod řádek}
    Tabulátor,        { označení začátku vodorovné tabulace }
    Mezera,           { označení začátku mezery }
    Hranice_Znaku,    { označení začátku každého znaku }
    Začátek_Slova,    { označení začátku slova }
    Začátek.Mezer);   { označení konce slova }

  Index_VP = (Základ, Přetisk);

  Popis_Vršku = množina_nad Popis;

  Půlpoložka = záznam
    Spodek: Osmice_Jehel;
    Vršek:  Popis_Vršku;
  Konec {Půlpoložka};

  Položka =
    pole [Index_VP] ze Půlpoložka;

Konstanta
  Mez_VP = 1024; { VP pro 1024 položek }

Proměnná
  Pomalý_Pohyb,              { příznak požadavku zpomalení }
  Zleva_Doprava: logický; { příznak jednosměrného tisku }
  První_Plná, {když VP není prázdná, tak index první položky}
  První_Prázdná, { index první volné položky ve VP }
  Levý_Okraj,
  Pravý_Okraj: celý;
  Zarovnání: bajt;           { 0..3, viz ESC a }

  Poloha_Hlavy: celý;        { kroků zleva k hlavě }
  Poloha_Balení: celý;       { kroků zleva za sbalený řádek }
  Nesbalené_Slovo: celý;     { index první nesbalené položky }
  Pomalu: logický;           { příznak pro zpomalení tisku }
```

```
  { Vyrovnávací paměť }
   VP: pole [0..Mez_VP] ze Položka;

{*********************************************************}

Procedura Zbytek_Řádku;
{ vymaže sbalenou část řádku z RVP a }
{       zbytek dá na začátek RVP      }
Proměnná
  i, j: celý;
Začátek
  j := Levý_Okraj;
  První_Plná := Levý_Okraj;
  Pro i := Nesbalené_Slovo vzestupně_do
      První_Prázdná - 1 opakuj
    VP [j] := VP [i];
    j := j + 1;
  Konec {Pro};
  První_Prázdná := j;
Konec {Zbytek_Řádku};

{*********************************************************}

Procedura Smaž_Všechno;
{ smaže celou RVP }
Začátek
  První_Prázdná := První_Plná;
Konec {Smaž_Všechno};

{*********************************************************}

Procedura Smaž_Znak;
{ smaže poslední znak v RVP zprava, maže i dříve vytištěné }
{ znaky mezi Levým_Okrajem a První_Plnou položkou }
Začátek
  Platí_li První_Prázdná > Levý_Okraj tak
    Odtud_opakuj { hledej, kde začíná znak }
      První_Prázdná := První_Prázdná - 1;
    až_bude_platit
      (Hranice_Znaku je_prvkem
      VP [První_Prázdná, Základ].Vršek)
      nebo (První_Prázdná = Levý_Okraj);
    Platí_li (První_Prázdná < První_Plná) tak
      První_Plná := První_Prázdná;
    Konec {Platí_li};
  Konec {Platí_li};
Konec {Smaž_Znak};

{*********************************************************}

Procedura Hlavu_Doleva;
{ tiskací hlava okamžitě najede na levý doraz }
Začátek
  OU.Vykonej_Posuny (-Poloha_Hlavy);
  Poloha_Hlavy := 0;
Konec {Hlavu_Doleva};

{*********************************************************}
```

```
Procedura Tiskni (Omezené_Zarovnání: bajt);
{ vytiskne obsah RVP a zarovná přitom tištěný řádek }
{ podle hodnoty parametru Omezené_Zarovnání         }
Proměnná
  Počet_Mezer,
  Prostrč_Každou_Mezeru,
  Zbývá_Prostrkat,
  L_Posun,
  P_Posun,
  Mez_Prostrku,
  L_Prostrk_Mezer,
  i: celý;
  P1,
  P2: Položka;
  U: RU.Popis_Úkonu;

{=============================================================}

Procedura Tisk_Sloupce_9 (X: Půlpoložka; P: Popis_Vršku);
{ vyšle úkon pro tisk vrchních 9 linek }
Začátek
  U.Spodní_Jehly := X.Spodek;
  U.Řízení := X.Vršek Typ RU.Řídicí_Signály
    * [RU.Jehla_9] + [RU.Svisle];
  OU.Zašli_Do_RU (U);
Konec {Tisk_Sloupce_9};

{*************************************************************}

Procedura Tisk_Sloupce_12 (X: Půlpoložka; P: Popis_Vršku);
{ vyšle úkon pro tisk 9. až 12. linky }
Začátek
  U.Spodní_Jehly := [];
  Platí_li Dvanáct_Linek je_prvkem P tak
    Platí_li J1 je_prvkem X.Spodek tak
      U.Spodní_Jehly := U.Spodní_Jehly + [J5, J7];
    Konec {Platí_li};
    Platí_li J2 je_prvkem X.Spodek tak
      U.Spodní_Jehly := U.Spodní_Jehly + [J6, J8];
    Konec {Platí_li};
  Konec {Platí_li};
  U.Řízení := [RU.Svisle];
  OU.Zašli_Do_RU (U);
Konec {Tisk_Sloupce_12};

{*************************************************************}

Procedura Udělej_Přetisk
 (Index_Do_VP: Index_VP;
  Nutné_Příznaky: Popis_Vršku;
  Procedura Tisk_Sloupce (X: Půlpolczka; P: Popis_Vršku));

{ vytiskne pomocí procedury Tisk_Sloupce všechny položky }
{ z RVP [Index_Do_VP], které obsahují Nutné_Příznaky     }
```

```
Proměnná
  L_Položka,          { levá krajní položka VP }
  P_Položka,          { pravá krajní položka VP }
  L_Poloha,           { levá krajní poloha hlavy }
  P_Poloha: celý; { pravá krajní poloha hlavy }
  i: celý;
  Zbývá: bajt;

{=========================================================}

Procedura Prostrkej_Mezery;
{ zvětšuje mezery o hodnotu Prostrč_Každou_Mezeru tak, }
{ aby se právě tištěný řádek zarovnal doleva i doprava }
Proměnná
  j: celý;
Začátek
  Tisk_Sloupce (VP [i, Index_Do_VP], VP [i, Základ].Vršek);
  Platí_li
      (Omezené_Zarovnání = 3) a_zároveň
      (i > L_Prostrk_Mezer) a_zároveň
      (Mezera je_prvkem VP[i, Základ].Vršek) tak
    Pro j := 1 vzestupně_do Prostrč_Každou_Mezeru +
        poř_č (Zbývá > 0) opakuj
      { opakuj 1. sloupec mezery }
      Tisk_Sloupce (VP [i, Index_Do_VP], VP [i, Základ].Vršek);
    Konec {Pro};
    Platí_li Zbývá > 0 tak
      Zbývá := Zbývá - 1;
    Konec {Platí_li};
  Konec {Platí_li};
Konec {Prostrkej_Mezery};

{*********************************************************}

Začátek {Udělej_Přetisk}
  Zbývá := Zbývá_Prostrkat;
  { najdi pravou a levou krajní položku, kterou vytiskneš }
  P_Položka := Poloha_Balení - 1;
  Dokud_platí
      (P_Položka >= První_Plná) a_zároveň neplatí
      ((Nutné_Příznaky <= VP [P_Položka, Základ].Vršek) a_zároveň
      ((Jehla_9 je_prvkem VP [P_Položka, Index_Do_VP].Vršek)
      nebo (VP [P_Položka, Index_Do_VP].Spodek <> [])))
      opakuj
    P_Položka := P_Položka - 1;
  Konec {Dokud_platí};
  L_Položka := První_Plná;
  Dokud_platí
      (L_Položka <= P_Položka) a_zároveň neplatí
      ((Nutné_Příznaky <= VP [L_Položka, Základ].Vršek) a_zároveň
      ((Jehla_9 je_prvkem VP [L_Položka, Index_Do_VP].Vršek)
       nebo (VP [L_Položka, Index_Do_VP].Spodek <> [])))
      opakuj
    L_Položka := L_Položka + 1;
  Konec {Dokud_platí};
```

```
  Platí_li L_Položka <= P_Položka tak
    { při zarovnání doleva i doprava tiskni všechny položky }
    Platí_li Omezené_Zarovnání = 3 tak
      L_Položka := První_Plná;
      P_Položka := Poloha_Balení - 1;
    Konec {Platí_li};
    L_Poloha := L_Položka + L_Posun;
    P_Poloha := P_Položka + P_Posun;
    U.Počet_Kroků := P_Poloha - L_Poloha + 1;
    Platí_li Pomalu tak
      U.Řízení := U.Řízení + [RU.Pomalu];
    Konec {Platí_li};
    { není-li třeba tisknout zleva, tiskni }
    { z té strany, ke které má hlava blíž  }
    Platí_li (abs (Poloha_Hlavy - L_Poloha) <=
        abs (Poloha_Hlavy - P_Poloha)) nebo
        RVP.Zleva_Doprava nebo Pomalu
        tak
      OU.Vykonej_Posuny (L_Poloha - Poloha_Hlavy);
      Poloha_Hlavy := P_Poloha + 1;
      U.Řízení := [RU.Vpřed, RU.Bouchej] + U.Řízení;
      Zakaž_Přerušení; { aby se nezdržovalo plnění fronty }
      RU.Přidej_Do_VP (U);
      Pro i := L_Položka vzestupně_do P_Položka opakuj
        Prostrkej_Mezery;
      Konec {Pro};
      { Obnov přerušení podle stavu fronty RZ. }
    jinak { zprava doleva }
      { stabilní poloha hlavy je na pravé straně kroku }
      L_Poloha := L_Poloha + 1;
      P_Poloha := P_Poloha + 1;
      OU.Vykonej_Posuny (P_Poloha - Poloha_Hlavy);
      Poloha_Hlavy := L_Poloha - 1;
      U.Řízení := [RU.Bouchej] + U.Řízení;
      Zakaž_Přerušení; { aby se nezdržovalo plnění fronty }
      RU.Přidej_Do_VP (U);
      Pro i := P_Položka sestupně_do L_Položka opakuj
        Prostrkej_Mezery;
      Konec {Pro};
      { Obnov přerušení podle stavu fronty RZ. }
    Konec {Platí_li};
  Konec {Platí_li};
Konec {Udělej_Přetisk};

{*********************************************************}

Začátek {Tiskni}
  { zjisti, zda se musí tisknout pomalu }
  Pro i := První_Plná vzestupně_do Poloha_Balení - 2 opakuj
    Platí_li neplatí Pomalu tak
      P1 := VP [i];
      P2 := VP [i + 1];
```

```
      Pomalu := Pomalý_Pohyb nebo
        (P1 [Základ].Spodek * P2 [Základ].Spodek
          <> []) nebo
        (P1 [Přetisk].Spodek * P2 [Přetisk].Spodek
          <> []) nebo
        (P1 [Základ].Vršek * P2 [Základ].Vršek
          * [Jehla_9] <> []) nebo
        (P1 [Přetisk].Vršek * P2 [Přetisk].Vršek
          * [Jehla_9] <> []);
    Konec {Platí_li};
  Konec {Pro};
  { připrav prostrkávání mezer pro zarovnání vlevo i vpravo }
  Mez_Prostrku := První_Plná;
  Dokud_platí (Mez_Prostrku < Poloha_Balení)
      a_zároveň neplatí (Začátek_Slova je_prvkem
      VP [Mez_Prostrku, Základ].Vršek)
      opakuj
    Mez_Prostrku := Mez_Prostrku + 1;
  Konec {Dokud_platí};
  L_Prostrk_Mezer := Poloha_Balení - 1;
  Počet_Mezer := 0;
  Dokud_platí (L_Prostrk_Mezer >= Mez_Prostrku) a_zároveň
      neplatí (Tabulátor je_prvkem
      VP [L_Prostrk_Mezer, Základ].Vršek)
      opakuj
    Počet_Mezer := Počet_Mezer + poř_č ((Mezera je_prvkem
      VP [L_Prostrk_Mezer, Základ].Vršek));
    L_Prostrk_Mezer := L_Prostrk_Mezer - 1;
  Konec {Dokud_platí};
  { spočítej posuny krajů řádku podle požadovaného zarovnání }
  P_Posun := Pravý_Okraj + 1 - Poloha_Balení;
  Vyber Omezené_Zarovnání ze
  0:
    Začátek
      L_Posun := 0;
      P_Posun := 0;
    Konec;
  1:
    Začátek
      P_Posun := P_Posun děl 2;
      L_Posun := P_Posun;
    Konec;
  2:
    Začátek
      L_Posun := P_Posun;
    Konec;
  3:
    Začátek
      L_Posun := 0;
      Platí_li Počet_Mezer = 0 tak
        Prostrč_Každou_Mezeru := 0;
        Zbývá_Prostrkat := 0;
      jinak
        Prostrč_Každou_Mezeru := P_Posun děl Počet_Mezer;
        Zbývá_Prostrkat := P_Posun mod Počet_Mezer;
      Konec {Platí_li};
    Konec;
  Konec {Vyber};
```

```
  { vytiskni řádek potřebným počtem průchodů hlavy }
  Udělej_Přetisk (Základ, [], Tisk_Sloupce_9);
  OU.Posuň_Papír_Na (OU.Poloha_Papíru + 3);
  Udělej_Přetisk (Přetisk, [], Tisk_Sloupce_9);
  OU.Posuň_Papír_Na (OU.Poloha_Papíru - 3 + 48);
  Udělej_Přetisk (Základ, [Dvanáct_Linek], Tisk_Sloupce_12);
  OU.Posuň_Papír_Na (OU.Poloha_Papíru + 3);
  Udělej_Přetisk (Přetisk, [Dvanáct_Linek], Tisk_Sloupce_12);
  OU.Posuň_Papír_Na (OU.Poloha_Papíru - 3 - 48);
  Pomalu := nepravda;
Konec {Tiskni};

{*************************************************************}

Procedura Doleva;
{ způsobí, že další znak se bude tisknout od levého okraje }
Začátek
  První_Prázdná := Levý_Okraj;
  První_Plná := Levý_Okraj;
Konec {Doleva};

{*************************************************************}

Procedura Vyprázdni_Přeplněnou;
{ sbalí a vytiskne řádku, zbytek řádky přesune doleva v RVP }
Proměnná
  i: celý;
Začátek
 { Nastav balení slov }
  Nesbalené_Slovo := První_Prázdná;
  Poloha_Balení := První_Prázdná;
  i := První_Prázdná - 1;
  Dokud_platí (i > První_Plná)
      {!Poloha_Balení > První_Plná, aby nebyl prázdný řádek!}
      a_zároveň (Poloha_Balení = První_Prázdná) opakuj
    Platí_li Začátek_Slova je_prvkem VP [i, Základ].Vršek tak
      Nesbalené_Slovo := i;
    Konec {Platí_li};
    Platí_li (Začátek.Mezer je_prvkem VP [i, Základ].Vršek)
        a_zároveň (i <= (Pravý_Okraj + 1)) tak
      Poloha_Balení := i;
    Konec {Platí_li};
    i := i - 1;
  Konec {Dokud_platí};

 {Vytiskni sbalenou část řádku a zbytek přesuň na začátek VP}
  Tiskni (Zarovnání);
  Platí_li Nesbalené_Slovo < Poloha_Balení
      { jen 1 slovo v řádku } tak
    Doleva;
  jinak
    Zbytek_Řádku;
  Konec {Platí_li};
Konec {Vyprázdni_Přeplněnou};

{*************************************************************}
```

```
Procedura Tiskni_A_Doleva;
Začátek
  Poloha_Balení := První_Prázdná;
  Platí_li Zarovnání > 2 tak
    Tiskni (0);
  jinak
    Tiskni (Zarovnání);
  Konec {Platí_li};
  Doleva;
Konec {Tiskni_A_Doleva};

{************************************************************}

Procedura Tiskni_Kus_Řádku;
Začátek
  Poloha_Balení := První_Prázdná;
  Tiskni (0);
  První_Plná := První_Prázdná; { smaž VP }
Konec {Tiskni_Kus_Řádku};

{************************************************************}

Procedura Přidej_Sloupec (X: Položka);
{ Uloží 1 položku do RVP }
Začátek
  Platí_li První_Prázdná <= Mez_VP tak
    VP [První_Prázdná] := X;
    První_Prázdná := První_Prázdná + 1;
  Konec {Platí_li};
Konec {Přidej_Sloupec};

{************************************************************}

Procedura Nastav;
{ počáteční nastavení proměnných modulu }
Začátek
  RVP.Zleva_Doprava := nepravda;
  Levý_Okraj := 0;
  Pravý_Okraj := Pravý_Doraz;
  Zarovnání := 0; { podle levého okraje }
  První_Plná := 0;
  První_Prázdná := 0;
  Pomalý_Pohyb := nepravda;
  Poloha_Hlavy := 0;
  Pomalu := nepravda;
Konec {Nastav};

{************************************************************}

Konec { RVP - řádková vyrovnávací paměť }.
```

```
Modul OU;
{ Ovladač úkonů }
{ Oprava z 06.06.88/10:45 }

Vývoz
  Poloha_Papíru, Svislé_Tabulátory, Kanál_Svislé_Tabulace,
  Výška_Stránky, Přeskok_Perforace, Rozteč_Řádků, Došel_Papír,
  Zvonek, Nezdar, Posuň_Papír_Na, Stránkuj, Řádkuj,
  Tabuluj_Svisle, Smaž_Tabulátory, Vykonej_Posuny,
  Zašli_Do_RU, Nastav;

Dovoz
  HW, RU, OU;

Konstanta
  Zarážek_V_Kanálu = 64;

Proměnná
  Poloha_Papíru: celý; { v 1/432" od začátku stránky }
  Svislé_Tabulátory: pole [0..7, 1..Zarážek_V_Kanálu] ze celý;
  Kanál_Svislé_Tabulace: bajt {0..7};
  Výška_Stránky: celý; { v 1/432" }
  Přeskok_Perforace: celý; { v 1/432" }
  Rozteč_Řádků: celý; { v 1/432" }

  { !!! hodnota proměnných Došel_Papír, Zvonek a Nezdar !!! }
  { !!!            bude předem pevně nastavena          !!! }
   { Strukturované konstanty pro úkony }
    Došel_Papír,
    Zvonek,
    Nezdar: RU.Popis_Úkonu;

  Kroky_Válce: celý;

{************************************************************}

Procedura Posuň_Papír_Na (X: celý { v 1/432" });
{ přičte do Kroky_Válce požadovaný posun v půllinkách }
Proměnná
  Y: celý;
Začátek
  Y := Poloha_Papíru;
  Poloha_Papíru := X mod Výška_Stránky;
 { přepočítej polohu X/432" na posun o X půllinek }
  { napřed spočítej do Y skut. polohu papíru v půllinkách }
  Y := (Y + 1) děl 3 * 3;
  Platí_li X < Y tak
    X := X - 2; { následující děl ufízne absolutní hodnotu }
  Konec {Platí_li};
  Kroky_Válce := Kroky_Válce + (X - Y + 1) děl 3;
 Konec {Posuň_Papír_Na};

{************************************************************}
```

```
Procedura Proveď_Svislé_Posuny;
{ Vykoná předem nastavený počet kroků válce }
Proměnná
  Úkon: RU.Popis_Úkonu;
Začátek
  Platí_li Kroky_Válce <> 0 tak
    Platí_li Kroky_Válce > 0 tak
      Úkon.Počet_Kroků := Kroky_Válce;
      Úkon.Řízení := Úkon.Řízení + [RU.Vpřed, RU.Svisle];
      RU.Přidej_Do_VP (Úkon);
    jinak
      Úkon.Počet_Kroků := -Kroky_Válce;
      Úkon.Řízení := Úkon.Řízení + [RU.Svisle];
      RU.Přidej_Do_VP (Úkon);
    Konec {Platí_li};
    Kroky_Válce := 0;
  Konec {Platí_li};
Konec {Proveď_Svislé_Posuny};

{************************************************************}

Procedura Vykonej_Posuny (Kroky_Hlavy: celý);
{ Vykoná předem nastavené svislé i zadané vodorovné posuny }
Proměnná
  Úkon: RU.Popis_Úkonu;
Začátek
  Proveď_Svislé_Posuny;
  Platí_li Kroky_Hlavy <> 0 tak
    Platí_li Kroky_Hlavy > 0 tak
      Úkon.Počet_Kroků := Kroky_Hlavy;
      Úkon.Řízení := Úkon.Řízení + [RU.Vpřed];
    jinak
      Úkon.Počet_Kroků := -Kroky_Hlavy;
    Konec {Platí_li};
    RU.Přidej_Do_VP (Úkon);
  Konec {Platí_li};
Konec {Vykonej_Posuny};

{************************************************************}

Procedura Stránkuj;
Začátek
  Posuň_Papír_Na (Výška_Stránky);
  Poloha_Papíru := 0;
  Proveď_Svislé_Posuny;
Konec {Stránkuj};

{************************************************************}
```

```
Procedura Řádkuj;
Proměnná
  Poloha_Řádku: celý;
Začátek
  Poloha_Řádku := Poloha_Papíru + Rozteč_Řádků;
  Platí_li Výška_Stránky - Přeskok_Perforace <
    Poloha_Řádku tak
    Stránkuj;
  jinak
    Posuň_Papír_Na (Poloha_Řádku);
    Proveď_Svislé_Posuny;
  Konec {Platí_li};
Konec {Řádkuj};

{************************************************************}

Procedura Tabuluj_Svisle;
Proměnná
  i: bajt;
  t,
  t2: celý;
Začátek
  i := 0; t := 0;
  Odtud_opakuj
    t2 := t;
    i := i + 1;
    t := Svislé_Tabulátory [Kanál_Svislé_Tabulace, i];
    Platí_li (t <= t2) nebo (i > Zarážek_V_Kanálu) tak
      t := Poloha_Papíru + Rozteč_Řádků;
    Konec {Platí_li};
  až_bude_platit t > Poloha_Papíru;
  Platí_li t > Výška_Stránky - Přeskok_Perforace tak
    Stránkuj;
  jinak
    Posuň_Papír_Na (t);
  Konec {Platí_li};
Konec {Tabuluj_Svisle};

{************************************************************}

Procedura Zašli_Do_RU (X: RU.Popis_Úkonu);
{ Tato procedura zajišťuje slučitelnost programů pro }
{       mechanické části tiskáren různých typů.       }
Začátek
  RU.Přidej_Do_VP (X);
Konec {Zašli_Do_RU};

{************************************************************}

Procedura Smaž_Tabulátory;
Proměnná
  i, j: bajt;
Začátek
  Pro i := 0 vzestupně_do 7 opakuj
    Pro j := 1 vzestupně_do Zarážek_V_Kanálu opakuj
      Svislé_Tabulátory [i, j] := 0;
    Konec {Pro};
  Konec {Pro};
```

```
Konec {Smaž_Tabulátory};

{************************************************************}

Procedura Nastav
  (Dvanáctipalcová_Stránka, Přeskakuj_Perforaci: logický);
{ počáteční nastavení proměnných modulu }
Proměnná
  i, j: bajt;
Začátek
  { !!! hodnota proměnných Došel_Papír, Zvonek a Nezdar !!! }
  { !!!            bude předem pevně nastavena          !!! }
    Došel_Papír.Vybraná_Písnička := RU.Došel_Papír;
    Došel_Papír.Řízení := [RU.Chyba, RU.Zahraj];
    Zvonek.Vybraná_Písnička := RU.Zvonek;
    Zvonek.Řízení := [RU.Zahraj];
    Nezdar.Vybraná_Písnička := RU.Nezdar;
    Nezdar.Řízení := [RU.Chyba, RU.Zahraj];

  Smaž_Tabulátory;
  Kanál_Svislé_Tabulace := 0;
  Platí_li Dvanáctipalcová_Stránka tak
    Výška_Stránky := 5184 + 72; { 12" }
  jinak
    Výška_Stránky := 4752 + 72; { 11" }
  Konec {Platí_li};
  Platí_li Přeskakuj_Perforaci tak
    Přeskok_Perforace := 432; { 1" }
  jinak
    Přeskok_Perforace := 0;
  Konec {Platí_li};
  Rozteč_Řádků := 72; { 1/6" }
  Kroky_Válce := 0;
  Poloha_Papíru := 0;
Konec {Nastav};

{************************************************************}

Konec { OU - ovladač úkonů }.
```

```
Modul RU;
{ Rozhraní úkonů }
{ Oprava z 04.06.88/09:20 }

Vývoz
  Význam_Písničky, Signál_K_Úkonu, Pomalu, Řídicí_Signály,
  Popis_Úkonu, Nastav, Prázdná_VP, Vyjmi_Z_VP, Přidej_Do_VP;

Dovoz
  HW, RU;

Typ
  Význam_Písničky =
   (Začínáme_Pracovat,
    Došel_Papír,
    Není_Doraz,
    Zvonek,
    Nezdar);

  Signál_K_Úkonu =
   (Jehla_9,
    Nic,
    Chyba,      { má se hlásit chyba }
    Vpřed,      { vpřed, jinak vzad }
    Svisle,     { svisle, jinak vodorovně }
    Bouchej,    { úder jehel, jinak nebouchej }
    Zahraj);    { má se zahrát písnička }

Konstanta
  Pomalu = Chyba; { v kombinaci s bouchej vodorovně   }
    { vpřed slouží jako požadavek na zpomalení pohybu hlavy }

Typ
  Řídicí_Signály = množina_nad Signál_K_Úkonu;

  Popis_Úkonu =
    záznam Vyber bajt ze
     0: (Vybraná_Písnička: Význam_Písničky);
     1: (Počet_Kroků: celý { mod 2048 });
     2: (Spodní_Jehly: Osmice_Jehel;
         Řízení: Řídicí_Signály);
    Konec {Popis_Úkonu};

Konstanta
  Mez_VP       = 255; { horní mez indexů ve vyr. paměti úkonů }

Proměnná
 { Vyrovnávací paměť }
  VP: pole [0..Mez_VP] ze Popis_Úkonu;
  První_Plná,          { index první nevyjmuté položky }
  První_Prázdná: bajt; { index první volné položky }

{*********************************************************)
```

```
Funkce Počet_Položek: bajt;
{ spočítá, kolik je obsazených položek ve VP }
Začátek
  Počet_Položek := (První_Prázdná - První_Plná) mod
    (Mez_VP + 1);
Konec {Počet_Položek};

{*********************************************************)

Procedura Přidej_Do_VP (X: Popis_Úkonu);
{ počká, až se uvolní VP, a vloží tam X }
Začátek
 { čekej na Počet_Položek < Mez_VP }
  Dokud_platí Počet_Položek >= Mez_VP opakuj
    { čekej }
  Konec {Dokud_platí};

 { Tady je ještě v První_Prázdná index první prázdné }
 { položky. Ve VP je volno, proto mohu uložit }
 { do první prázdné položky. }
  VP [První_Prázdná] := X;
 { ! První_Prázdná teď neukazuje na prázdnou položku ! }
  První_Prázdná := (První_Prázdná + 1) mod (Mez_VP + 1);
 { Tady už je zase První_Prázdná v pořádku. }
Konec {Přidej_Do_VP};

{*********************************************************)

Funkce Prázdná_VP: logický;
{ říká, že VP je prázdná a nelze volat Vyjmi_Z_VP }
Začátek
  { Když VP obsahuje aspoň 1 položku, lze zavolat }
  { Vyjmi_Z_VP, jinak je třeba považovat VP za prázdnou. }
  Prázdná_VP := Počet_Položek = 0;
Konec {Prázdná_VP};

{*********************************************************)

Procedura Vyjmi_Z_VP (X: word { Proměnná X: Popis_Úkonu });
{ Když je Prázdná_VP, tak se pomate }
Začátek
 { Nyní bude porušen význam proměnné První_Plná }
  X := VP [První_Plná];
  První_Plná := (První_Plná + 1) mod (Mez_VP + 1);
 { tady je už zase První_Plná v pořádku }
Konec {Vyjmi_Z_VP};

{*********************************************************)

Procedura Nastav;
{ počáteční nastavení VP před prvním použitím VP }
Začátek
  První_Plná := 0;
  První_Prázdná := 0;
Konec {RU_Nastav_VP};

{*********************************************************)

Konec {RU - Rozhraní úkonů}.
```

```
Modul RZ;
{ Rozhraní znaků }
{ Oprava z 04.06.88/09:15 }

Vývoz
  Nastav, Je_Znak_Ve_VP, Vyjmi_Z_VP, INTSVC;

Dovoz
  HW;

Konstanta
  Hotovo      =  S4; { rozsvit světýlko READY }
  NACK        =  S5; { signál ACK na rozhraní Centronics }

  Mez_VP      = 255; { horní mez indexů ve vyr. paměti znaků }

Proměnná
 { Vyrovnávací paměť }
  VP: pole [0..Mez_VP] ze bajt;
  První_Plná,          { index první nevyjmuté položky }
  První_Prázdná: bajt; { index první volné položky }

{*********************************************************}

Funkce Počet_Položek: bajt;
{ spočítá, kolik je obsazených položek ve VP }
Začátek
  Počet_Položek := (První_Prázdná - První_Plná) mod
    (Mez_VP + 1);
Konec {Počet_Položek};

{*********************************************************}

Procedura INTSVC;
{ obsluha přerušení INT při příchodu dat na port HW.Data }
Začátek
  { automaticky se zakázalo přerušení }
  { uschovej stav procesoru, např. v instrukcích 8080: }
    push PSW
    push B
    push D
    push H;

 { čekej na Počet_Položek >= Mez_VP }
  Platí_li Počet_Položek >= Mez_VP tak
    HW.Řízení_8255 := 2 * poř_č (Hotovo) + 0;
    { obnov stav procesoru, např. v instrukcích 8080: }
      pop H
      pop D
      pop B
      pop PSW;
    { přerušení zůstane zakázané, dokud se neuvolní VP }
  jinak

   { Tady je ještě v První_Prázdná index první prázdné }
   { položky. Ve VP je volno, proto mohu uložit }
   { do první prázdné položky. }
    VP [První_Prázdná] := HW.Data;
   { ! První_Prázdná teď neukazuje na prázdnou položku ! }
```

```
      První_Prázdná := (První_Prázdná + 1) mod (Mez_VP + 1);
     { Tady už je zase První_Prázdná v pořádku. }

      HW.Řízení_8255 := 2 * poř_č (NACK) + 0;
      { prodloužení pulsu NACK na 12 us }
      HW.Řízení_8255 := 2 * poř_č (NACK) + 1;

      { obnov stav procesoru, např. v instrukcích 8080: }
        pop H
        pop D
        pop B
        pop PSW;
      Povol_Přerušení;
   Konec {Platí_li};
Konec {INTSVC};

{*********************************************************}

Funkce Je_Znak_Ve_VP: logický;
{ říká, že VP není prázdná a lze volat Vyjmi_Z_VP }
Začátek
   { Když   VP   obsahuje   aspoň  1  položku,  lze  zavolat }
   { Vyjmi_Z_VP, jinak je třeba považovat VP za prázdnou. }
   Je_Znak_Ve_VP := Počet_Položek <> 0;
Konec {Je_Znak_Ve_VP};

{*********************************************************}

Funkce Vyjmi_Z_VP: bajt;
{ Když je prázdná VP, tak se pomate }
Začátek
  { Nyní bude porušen význam proměnné První_Plná }
   Vyjmi_Z_VP := VP [První_Plná];
  { Mezi změnou První_Plná a povolením přerušení se nesmí    }
  { přerušit, aby se náhodným přerušením nenarušil vztah     }
  { mezi počtem položek ve VP a zákazem/povolením přerušení }
   Zakaž_Přerušení;
   První_Plná := (První_Plná + 1) mod (Mez_VP + 1);
  { tady je už zase První_Plná v pořádku }
   HW.Řízení_8255 := 2 * poř_č (Hotovo) + 1;
  { VP nemůže být plná, lze přerušovat, ale ne z NMISVC }
   Povol_Přerušení; { umožní přidávání dalších položek }
Konec {Vyjmi_Z_VP};

{*********************************************************}

Procedura Nastav;
{ počáteční nastavení VP před prvním použitím VP }
Začátek
   První_Plná := 0;
   První_Prázdná := 0;
Konec {RU_Nastav_VP};

{*********************************************************}

Konec {RZ - Rozhraní znaků}.
```

```
Modul OM;
{ Ovladač mechaniky - motorů, hlavy a pípáku }
{ Oprava z 09.06.88/06:50 }

{ ! Procedura Konej se musí vždy vzdát pro- ! }
{ ! cesoru před příchodem následujícího NMI ! }

{ pozn.: zkratka "us" znamená "mikrosekunda" }

Vývoz
  Nastav_Proces, Pozastavený;

Dovoz
  HW, RU, OP;

Konstanta
  Levý_Doraz   = S0; { tiskací hlava je na levém dorazu }
  Je_Napětí    = S6; { motory jsou napájeny }

  J9           = S0; { spodní jehla v hlavičce }
  Barvi        = S2; { převíjej barvicí pásku }
  Vše_V_Pořádku = S3; { signál ERR na rozhraní Centornics }

  Pomalu       = 3200  {us}; { na 1 krok hlavy }
  Rychle       = 1600  {us}; { na 1 krok hlavy }
  Perioda_Válce = 10000 {us}; { na 1 krok válce }
  Dno          = 64;   { bytů zásobníku }

 { konstanty pro rozjezdy a brždění hlavy }
  Kroků_Rozjezdu = 19;
  R1 = 2400 {us};
  B1 = 2400 {us};

Proměnná
  Perioda_Hlavy: celý {us};
  Hlava_Vpravo: logický; { odchylka dobrzděné hlavy vůči }
                         { předpokládané poloze }
  PU: RU.Popis_Úkonu;
  Pozastavený: OP.Proces;
  Zásobník: pole [1..Dno] ze bajt;

{************************************************************}

Funkce Počet_Kroků: celý;
{ Vyjme z PU.Počet_Kroků 10 nejméně významných bitů }
Začátek
  Počet_Kroků := PU.Počet_Kroků mod 2048;
Konec {Počet_Kroků};

{************************************************************}

Procedura Perioda (P: celý);
{ nastaví periodu hodin v us a počká na NMI }
Začátek
  HW.Hodiny := Low (P);
  HW.Hodiny := High (P);
  Transfer (Pozastavený, Pozastavený);
Konec {Perioda};
```

```
{*********************************************************}

Procedura Zahraj_Písničku (P: RU.Význam_Písničky);
Proměnná
  i: bajt;
  { !!! hodnota proměnné Písnička   !!! }
  { !!! bude předem pevně nastavena !!! }
  Písnička:
    pole [RU.Význam_Písničky, 0..3] ze
      záznam
        Výška,       { perioda v us }
        Délka: celý; { v ms }
      Konec {záznam};

  Procedura Nastavení_Písniček;
  { Pevné nastavení hodnot do proměnné Písnička, }
  { tato procedura se nebude nikdy volat         }
  Začátek
    { !!! hodnota proměnné Písnička   !!! }
    { !!! bude předem pevně nastavena !!! }
    Písnička [RU.Začínáme_Pracovat, 0].Výška := 1933;
    Písnička [RU.Začínáme_Pracovat, 0].Délka := 750;
    Písnička [RU.Začínáme_Pracovat, 1].Výška := 2048;
    Písnička [RU.Začínáme_Pracovat, 1].Délka := 250;
    Písnička [RU.Začínáme_Pracovat, 2].Výška := 1722;
    Písnička [RU.Začínáme_Pracovat, 2].Délka := 250;
    Písnička [RU.Začínáme_Pracovat, 3].Výška := 1933;
    Písnička [RU.Začínáme_Pracovat, 3].Délka := 250;
    Písnička [RU.Došel_Papír, 0].Výška := 1534;
    Písnička [RU.Došel_Papír, 0].Délka := 375;
    Písnička [RU.Došel_Papír, 1].Výška := 1722;
    Písnička [RU.Došel_Papír, 1].Délka := 125;
    Písnička [RU.Došel_Papír, 2].Výška := 1933;
    Písnička [RU.Došel_Papír, 2].Délka := 1000;
    Písnička [RU.Došel_Papír, 3].Výška := 2580;
    Písnička [RU.Došel_Papír, 3].Délka := 500;
    Písnička [RU.Není_Doraz, 0].Výška := 2580;
    Písnička [RU.Není_Doraz, 0].Délka := 500;
    Písnička [RU.Není_Doraz, 1].Výška := 2734;
    Písnička [RU.Není_Doraz, 1].Délka := 250;
    Písnička [RU.Není_Doraz, 2].Výška := 2580;
    Písnička [RU.Není_Doraz, 2].Délka := 250;
    Písnička [RU.Není_Doraz, 3].Výška := 1534;
    Písnička [RU.Není_Doraz, 3].Délka := 1000;
    Písnička [RU.Zvonek, 0].Výška := 2299;
    Písnička [RU.Zvonek, 0].Délka := 1000;
    Písnička [RU.Zvonek, 1].Výška := 1722;
    Písnička [RU.Zvonek, 1].Délka := 500;
    Písnička [RU.Zvonek, 2].Výška := 1933;
    Písnička [RU.Zvonek, 2].Délka := 500;
    Písnička [RU.Zvonek, 3].Výška := 2048;
    Písnička [RU.Zvonek, 3].Délka := 500;
    Písnička [RU.Nezdar, 0].Výška := 1722;
    Písnička [RU.Nezdar, 0].Délka := 250;
    Písnička [RU.Nezdar, 1].Výška := 1933;
    Písnička [RU.Nezdar, 1].Délka := 250;
    Písnička [RU.Nezdar, 2].Výška := 2048;
    Písnička [RU.Nezdar, 2].Délka := 500;
    Písnička [RU.Nezdar, 3].Výška := 1722;
```

```
    Písnička [RU.Nezdar, 3].Délka := 500;
  Konec {Nastavení_Písniček};

Začátek {Zahraj_Písničku}
  Perioda (1000 {us});
  { Začátek cyklu } i := 0; Odtud_opakuj
    { pozn.: Low = méně významný bajt, }
    {        High = významnější bajt   }.
    HW.Výška_Tónu := Low  (Písnička [P, i].Výška);
    HW.Výška_Tónu := High (Písnička [P, i].Výška);
    HW.Délka_Tónu := Low  (Písnička [P, i].Délka);
    HW.Délka_Tónu := High (Písnička [P, i].Délka);

    { Pozor ! Předvolba časovače se nastaví až náběžnou }
    { a sestupnou hranou hodin, pak teprve mohu číst stav }
    { čítače. Protože Perioda čeká na sestupnou a náběžnou }
    { hranu hodin, musím ji zavolat dvakrát. }
    Perioda (1000 {us});
    { Teď přišla poprvé náběžná hrana hodin. }
    Odtud_opakuj
      Perioda (1000 {us});
      { Při prvním průchodu cyklem se sestupnou hranou }
      { hodin teprve nastaví předvolba do čítače. }
    až_bude_platit (HW.Délka_Tónu {LSB} Typ Osmice_Signálů
        + HW.Délka_Tónu {MSB} Typ Osmice_Signálů = []);
     { pozn.: LSB = méně význ. bajt, MSB = významnější bajt }
  { Konec cyklu } i := i + 1; až_bude_platit i > 3;
Konec {Zahraj_Písničku};

{************************************************************}

Procedura Ohlaš_Chybu (P: RU.Význam_Písničky);
{ nuluje signál ERR na rozhraní Centronics }
{ a zahraje písničku }
Začátek
  HW.Řízení_8255 := 2 * poř_č (Vše_V_Pořádku) + 0;
  Zahraj_Písničku (P);
Konec {Ohlaš_Chybu};

{************************************************************}

Procedura Najeď_Doleva;
{ najede hlavou na fyzický levý kraj papíru kousek od dorazu }
Návěští 1;
Konstanta
  Šířka_Válce = 1320; { kroků motoru max. }
  Kousek_Vpravo = 30; { kroků od levého dorazu: [1..256] }
Proměnná
  i: celý; { řídicí proměnná cyklu }

Začátek {Najeď_Doleva}
 { Podívej se, zda jsou napájeny motory }
  Dokud_platí neplatí (Je_Napětí je_prvkem HW.Stav) opakuj
    Ohlaš_Chybu (RU.Není_Doraz);
  Konec {Dokud_platí};
```

```
  { Je-li hlava na levém dorazu, popojeď doprava }
   Platí_li Levý_Doraz je_prvkem HW.Stav tak
     Opakuj Kousek_Vpravo krát
       Perioda (R1);
       Vykonej_Krok (HW.Vpravo);
     Konec {Opakuj};
     Dokud_platí Levý_Doraz je_prvkem HW.Stav opakuj
       Ohlaš_Chybu (RU.Není_Doraz);
     Konec {Dokud_platí};
   Konec {Platí_li};
  { Pokus se najet na levý kraj }
   Odtud_opakuj
     { Začátek cyklu } i := Šířka_Válce; Odtud_opakuj
       Perioda (R1);
       Platí_li Levý_Doraz je_prvkem HW.Stav tak
         Pokračuj_na_návěští 1;
       Konec {Platí_li};
       Vykonej_Krok (HW.Vlevo);
     { Konec cyklu } i := i - 1; až_bude_platit i = 0;
     Odtud_opakuj
       Ohlaš_Chybu (RU.Není_Doraz);
     až_bude_platit Levý_Doraz je_prvkem HW.Stav;
     1: { sem skoč, když najdeš doraz }
     Perioda (50000);
   až_bude_platit Levý_Doraz je_prvkem HW.Stav;
  { Ještě i po 50 ms je hlava na levém dorazu. }

  { Nastav pevnou polohu papíru }
   Perioda (Perioda_Válce);
   Vykonej_Krok (HW.Dolů);
   Perioda (1000); { Nastav velmi krátkou periodu }

   HW.Řízení_8255 := 2 * poř_č (Vše_V_Pořádku) + 1;
   Hlava_Vpravo := nepravda;
Konec {Najeď_Doleva};

{*********************************************************}

Procedura Rozjeď (Směr: bajt { adresa portu });
Začátek
   Perioda (R1); { počkej, až se ustálí hlava }
   { kroky rozjezdu }
   Perioda (R1); { perioda pro následující krok }
   Vykonej_Krok (Směr); { krok motoru }
   Opakuj Kroků_Rozjezdu - 2 krát
     Perioda (Perioda_Hlavy);
   Vykonej_Krok (Směr); { krok motoru }
   Konec {Opakuj};
Konec {Rozjeď};

{*********************************************************}

Procedura Brzdi (Směr: bajt { adresa portu });
Začátek
   Opakuj Kroků_Rozjezdu krát
     Perioda (B1);
   Vykonej_Krok (Směr); { krok motoru }
   Konec {Opakuj};
Konec {Brzdi};
```

```
{*********************************************************}

Procedura Vodorovně (Bouchej: logický; Směr: bajt {adr. portu});
{ Rozjede hlavu požadovaným směrem a posune ji o počet }
{ kroků obsažený při vyvolání procedury v PU. Je-li }
{ Bouchej, bude se při každém kroku číst kombinace jehel }
{ z RU a tisknout. }
Proměnná
  i: celý; { řídicí proměnná cyklu }
Začátek
  Perioda (1000); { jinak by se mohl přešvihnout čas }
  { rozjeď tiskací hlavu, před tím ji podle potřeby posuň }
  Perioda_Hlavy := Rychle;
  i := Počet_Kroků;
  Platí_li i < 2 * Kroků_Rozjezdu nebo Bouchej tak
    Platí_li Hlava_Vpravo tak
      Platí_li Směr = HW.Vpravo tak
        Rozjeď (HW.Vlevo);
        Brzdi (HW.Vlevo);
      Konec {Platí_li};
    jinak
      Platí_li Směr = HW.Vlevo tak
        Rozjeď (HW.Vpravo);
        Brzdi (HW.Vpravo);
      Konec {Platí_li};
    Konec {Platí_li};
    Hlava_Vpravo := Směr = HW.Vpravo;
  jinak
    i := i - 2 * Kroků_Rozjezdu;
    { výsledná odchylka hlavy se nezmění }
  Konec {Platí_li};

  Platí_li Bouchej a_zároveň (RU.Pomalu je_prvkem PU.Řízení) tak
    Perioda_Hlavy := Pomalu;
  { jinak nech Perioda_Hlavy = Rychle }
  Konec {Platí_li};
  Rozjeď (Směr);
  Platí_li i <> 0 tak

    { ošetři pohyb barvicí pásky }
    Platí_li neplatí (RU.Pomalu je_prvkem PU.Řízení) tak
      HW.Řízení_8255 := 2 * poř_č (Barvi) + 1;
        { převíjej barvicí pásku }
    Konec {Platí_li};

    { pohni s hlavou o požadovaný počet kroků a příp. tiskni }
    { začátek cyklu, !!! i = Počet_Kroků !!! } Odtud_opakuj
      Platí_li Bouchej a_zároveň neplatí RU.Prázdná_VP tak
        RU.Vyjmi_Z_VP (PU);
      Konec {Platí_li};
      { je-li prázdná VP, ponechá se v RU stará hodnota; }
      { přebytečné kombinace jehel pak musí obsahovat    }
      { úkon zpětný posun papíru }
      Perioda (Perioda_Hlavy);
      Vykonej_Krok (Směr); { krok motoru }
```

```
      Platí_li Bouchej tak
        HW.Spodní_Jehly := PU.Spodní_Jehly;
        Platí_li RU.Jehla_9 je_prvkem PU.Řízení tak
          HW.Řízení_8255 := 2 * poř_č (J9) + 1;
        jinak
          HW.Řízení_8255 := 2 * poř_č (J9) + 0;
        Konec {Platí_li};
        { počkej asi 50 us, aby sepnuly spínače jehel }
        Vystup_Na_Port (HW.Bum);
      Konec {Platí_li};
    { konec cyklu } i := i - 1; až_bude_platit i = 0;
    HW.Řízení_8255 := 2 * poř_č (Barvi) + 0;
      { zastav barvicí pásku }
  Konec {Platí_li};
  Brzdi (Směr);
Konec {Vodorovně};

{*********************************************************}

Procedura Konej;
{ realizuje paralelní proces ovládání mechaniky - }
{ opakovaně čte popisy úkonů a vykonává je         }
Proměnná
  i: celý; { řídicí proměnná cyklu }
Začátek
  Najeď_Doleva;
  Zahraj_Písničku (RU.Začínáme_Pracovat);
  Odtud_opakuj
    Dokud_platí RU.Prázdná_VP opakuj
      Perioda (1000 {us});
    Konec {Dokud_platí};
    RU.Vyjmi_Z_VP (PU);
    Platí_li RU.Zahraj je_prvkem PU.Řízení tak
      Platí_li RU.Chyba je_prvkem PU.Řízení tak
        Ohlaš_Chybu (PU.Vybraná_Písnička);
      jinak
        Zahraj_Písničku (PU.Vybraná_Písnička);
      Konec {Platí_li};
    jinak
      HW.Řízení_8255 := 2 * poř_č (Vše_V_Pořádku) + 1;
      Platí_li RU.Svisle je_prvkem PU.Řízení tak
        Platí_li RU.Vpřed je_prvkem PU.Řízení tak
          { Začátek cyklu } i := Počet_Kroků; Odtud_opakuj
            Perioda (Perioda_Válce);
            Vykonej_Krok (HW.Dolů);
          { Konec cyklu } i := i - 1; až_bude_platit i = 0;
        jinak { svisle vzad }
          { Začátek cyklu } i := Počet_Kroků; Odtud_opakuj
            Perioda (Perioda_Válce);
            Vykonej_Krok (HW.Nahoru);
          { Konec cyklu } i := i - 1; až_bude_platit i = 0;
        Konec {Platí_li};
      jinak { vodorovně }
        Platí_li RU.Vpřed je_prvkem PU.Řízení tak
          Vodorovně (RU.Bouchej je_prvkem PU.Řízení, HW.Vpravo);
        jinak { vodorovně vzad }
          Vodorovně (RU.Bouchej je_prvkem PU.Řízení, HW.Vlevo);
        Konec {Platí_li};
      Konec {Platí_li};
```

```
    Konec {Platí_li};
  až_bude_platit nepravda; { opakuj pořád dokola }
Konec {Konej};

{************************************************************}

Procedura Nastav_Proces;
{ Nastaví Zásobník a Pozastavený před spuštěním Konej }
Začátek
  { Tato procedura je závislá na technickém vybavení a na }
  { použitém programovacím jazyce. Zajistí, aby procedura }
  { Transfer předala procesor procesu, který začne vykoná-}
  { vat proceduru Konej. }
  Zásobník [Dno-1] Typ adresa := adresa_na (Konej);
  Pozastavený := adresa_na (Zásobník [Dno-1]);
Konec {Nastav_Proces};

{************************************************************}

Konec {OM - Ovladač mechaniky}.
```

```
Modul OP;
{ Ovladač procesů a hlavní program pro řízení tiskárny }
{ Oprava z 01.06.88/13:55 }

Vývoz
  Transfer, NMISVC, Tisk;

Dovoz
  HW, OM;

Typ
  proces = adresa;

Konstanta
  Netikej = S6;

{************************************************************}

Procedura Transfer (Proměnná P1, P2: proces);
{ Proces P1 předá procesor procesu P2:                       }
{ vyjme obsah P2, do P1 vloží SP-registr právě běží-         }
{ cího procesu a SP-reg. naplní původním obsahem P2.         }
Začátek
  { v instrukcích procesoru 8080: }
  mov H, B
  mov L, C
  mov C, M
  inx H
  mov B, M
  lxi H, 0
  dad SP
  xchg
  mov M, E
  inx H
  mov M, D
  mov H, B
  mov L, C
  sphl
  ret;
Konec {Transfer};

{************************************************************}

Procedura NMISVC;
Začátek
  { uschovej registry - např. v instrukcích Z80: }
    ex AF, AF'
    exx;
  Transfer (OM.Pozastavený, OM.Pozastavený);
  { obnov obsah registrů - např. v instrukcích Z80: }
    exx
    ex AF, AF';
  { vrať procesor přerušenému procesu - např. v instr. Z80: }
    retn;
Konec {NMISVC};

{************************************************************}
```

```
Procedura Tisk;
{ Tato procedura bude spuštěna technickým     }
{ vybavením tiskárny hned po zapnutí do sítě }
Začátek {*** začátek hlavního programu ***}
  { je maskováno NMI }

 { počáteční nastavení časovače 8253 }
  HW.Řízení_8253 := 034h;
    { hodiny pracují v módu 2 - NMI každou 1 ms }
  { pozn.: Low = méně význ. bajt, High = významnější bajt }
  HW.Hodiny := Low (1000);
  HW.Hodiny := High (1000); { po dobu 1 ms nebude NMI }
  HW.Řízení_8253 := 076h;
    { signál pro pípák se tvoří v módu 3 }
  HW.Výška_Tónu := 1; HW.Výška_Tónu := 0;
    { neslyšitelný tón }
  HW.Řízení_8253 := 0B0h;
    { propouštění a maskování tónu v módu 0 }
  HW.Délka_Tónu := 1; HW.Délka_Tónu := 0;
    { co nejdříve maskuj tón }

 { počáteční nastavení vstupního a výstupního obvodu 8255 }
  HW.Řízení_8255 := 90h; { tím se může odmaskovat NMI }
  HW.Řízení_8255 := 2 * poř_č (Netikej) + 1; { maskuj NMI }

  { !!! až do této chvíle nesmí uplynout víc než 1 ms !!! }
  { !!! od nastavení hodin                            !!! }

 { počáteční nastavení obsahu zásobníků pro procesy }
  OM.Nastav_Proces;

 { počáteční nastavení front }
  RU.Nastav;
  RZ.Nastav;

 { spuštění procesů }
  Povol_Přerušení;
  { povol - odmaskuj NMI, na rozhraní CENTRONICS nastav }
  { ERR, ACK, READY }
  HW.Řízení := [S3, S4, S5];
  ZZ.Zpracuj_Znak (HW.Přepínače_Předvolby);

Konec {*** konec hlavního programu ***};

Konec { OP - ovladač procesů }.
```

6. O práci ladiče programů

Hoj, trpělivý čtenáři, dobral ses konce našeho křivolakého putování od zadání úlohy k návrhu jejího řešení. Zastav již své kroky na cestě programování a dav spočinouti znaveným údům, popatři ještě spolu se spisovatelem na cestu ubíhající vpřed do nedozírných dálav změtí programovacích jazyků, rozličných procesorů, počítačů a systémových programů. Vyprovoďme již jen letmým pohledem až dosud hýčkaný výplod našich myslí, jenž by se měl v tomto okamžiku vydat cestami praktického uskutečnění, cestami plnými střetů našich představ s objektivní skutečností.

Nastiňme aspoň stručně další cestu našeho programu. Na první pohled se zdá být jednoduchá: náš program musí být zapsán v nějakém programovacím jazyce a z této podoby přeložen pomocí vhodného programu na vhodném počítači do instrukcí toho procesoru, který je použit v řídicím počítači tiskárny. Možností je mnoho a volba není snadná. Navíc se mohou názory různých lidí zásadně lišit. V těchto věcech je třeba volit podle vlastních schopností, možností a zkušeností. Jsou věci, do kterých si programátor nemá nechat mluvit, protože když udělá tu chybu, že si dá poradit, zpravidla pak toho trpce lituje. Pisatel se necítí povolán k tomu, aby na tomto místě předložil čtenářské obci výběr výše zmíněných prostředků a podrobně diskutoval jejich výhody a nevýhody pro naprogramování naší úlohy. Pisatel si dovoluje upozornit nebohého čtenáře, aby si prozatím nezoufal nad svou bezmocností - skutečnost je ještě daleko horší.

Je bláhové se domnívat, že přepsáním programu do vybraného jazyka bude úloha vyřešena. Tím teprve začíná klopotná cesta napravování chyb starých a páchání chyb nových. Okřídlená poučka totiž praví, že každý program obsahuje aspoň jednu chybu. Pokud se zdá, že v programu žádná chyba není, jedná se zřejmě o chybu skrytou, tím i záludnou, a tudíž mimořádně nebezpečnou. Druhá poučka praví, že každá oprava vnáší do programu aspoň jednu chybu. Úkolem programátora je omezit chyby v programu aspoň do té míry, aby se o nich nikdy nikdo nedověděl, aby o nich nevěděl ani sám programátor, aby zákazník i uživatel byli uchlácholeni iluzí, že v programu žádné chyby nejsou. Zkoušení programu a odstraňování chyb se nazývá **ladění**. Při ladění se program ze zdánlivě nevinného výmyslu stává nebezpečným běsem jednajícím nezávisle na vůli programátorově, běsem, který ohrožuje nejen použitou výpočetní techniku, nýbrž i duševní zdraví svého tvůrce. Při ladění programátor zápasí sám se sebou (přesněji řečeno: s temnými místy ve své minulosti). Na základě praktických neúspěchů musí pátrat ve svých myšlenkových pochodech po chybách, musí líčit logické pasti na své vlastní omyly. Počet chyb lze omezit vlastně jenom dvěma způsoby: v horším případě se programátor musí poučit z chyb vlastních, v lepším případě se dokáže poučit z chyb cizích. I povrchní pozorovatel zaznamená, že

programátor je bytost obdařená samorostlým duševnem. Nevyřešenou hádankou však zůstává, zda programátorská samorostlost je nezbytnou vrozenou vlohou k programování, nebo je naopak důsledkem povolání programátora.

V zájmu zachování svého životního štěstí užívají programátoři několika způsobů, jak učinit z ladění programu příjemnou zábavu, jako teplý večerní vánek zlehka šimrající zteřelou šedou kůru mozkovou. S jednou takovou metodou jsme se již seznámili: struktura programu je navržena tak, aby napomáhala odhalení chyb a určení jejich příčin. Dalším způsobem je používání takových prostředků pro ladění, které umožňují snadno chyby odhalovat a rychle je opravovat. Jakých prostředků však můžeme použít my, když naší tiskárně slouží k dorozumnění s obsluhou všeho všudy dvě tlačítka a dvě světýlka? Jediným rozumným řešením je ladit program pomocí jiného počítače. Tomu napomáhá i zvolená modulární struktura.

Modul RZ můžeme nahradit standardním čtením znaku z klávesnice nebo z magnetického disku a modul RU kontrolním výpisem na displej. Pouze modul OM a mechanismy sdílení procesoru několika procesy budeme muset odladit přímo v řídicím počítači tiskárny, protože na jiném počítači pravděpodobně nebude odpovídající technické vybavení. Přinejmenším však můžeme v řídicím počítači pro začátek vynechat moduly RZ, RP, ZZ, GZ, RVP a OU. Toho dosáhneme tak, že modul RU nahradíme modulem, který bude přijímat pokyny k provedení úkonů z rozhraní CENTRONICS. V nadřízeném počítači pak může být umístěn buď zjednodušený vysílač úkonů, nebo (v závěrečné fázi ladění) ostatní předem odladěné moduly. Takto odladěný program se pak trvale umístí do paměti řídicího počítače tiskárny.

Na okraj poznamenejme, že rozdělení programu mezi dva procesory používá i firma EPSON u jedné ze svých novějších tiskáren z podzimu 1987: jeden procesor přijímá znaky z nadřízeného počítače a zpracovává je na pokyny pro ovladač mechaniky. Na druhém procesoru pak běží ovladač mechaniky.

Na závěr této kapitoly autor znovu zdůrazňuje, že bylo jeho cílem poskytnout čtenáři jen letmý pohled na další osudy řádně zadaného a podrobně rozebraného programu. Zvídavý čtenář to může považovat za námět pro hlubší studium programování, zatímco ostatním čtenářům poslouží tato kapitola jako doplňující informace k vyložené látce.

7. Na rozloučenou se čtenářem

Autorovi nyní nezbývá než vyjádřit svůj obdiv k trpělivosti laskavého čtenáře, pokud dočetl Veselou rukověť až na toto místo. Pokud snad soudný čtenář kypěl při čtení tohoto pojednání nesouhlasem s obsahem či formou a nechal snad i dopadnout ostří své kritiky na hlavu ješitného spisovatele, nechť nyní odloží veškerou trpkost, neboť bylo cílem autorovým spíše podnítit samostatné úvahy čtenářů a probudit v nich zájem, než předložit čtenářské obci pochybný souhrn pouček o programování.

Autor si dovoluje touto cestou předem vyjádřit svůj dík těm čtenářům, kteří mu sdělí své dojmy, kritické připomínky a náměty na zdokonalení Veselé rukověti programování. Veselá rukověť by nemohla vzniknout bez podpory celé řady spolupracovníků, kteří pomáhali autorovi radou i skutkem v oblasti odborné, metodické, organizační a jazykové. Jmenujme alespoň autorku obrázků Ing. Michaelu Roedlovou. Autor děkuje všem, kdo přispěli svou hřivnou ke zdaru Veselé rukověti programování.

Veselá rukověť programování. Autor Ing. Ivan Ryant. Lektoroval Ing. Josef Truxa. Členská knižnice 602. základní organizace Svazarmu, Dr. Z. Wintra 8, 160 41 Praha 6. Odpovědný redaktor PhDr. Jindřich Jirka. Návrh obálky a grafické řešení Josef Svoboda. Technická redakce Daniela Prokopová. Vytiskly Tiskařské závody Praha, provoz Čelákovice.
Povoleno MK ČSR 1989.

Cena 33,-Kčs