Klub, který zná svůj konec?

Kdy se u nás objevily první sinclairy? Datum je samozřejmě nemožné určit, ale prvních padesát vyznavačů jednoduchého osobního počítače označeného jako ZX 81 se začalo scházet již v roce 1981. Technický pokrok, zejména pak dostupnost počítače s barevným vybavením a vynikající zobrazovací schopností jakou má Spektrum, vyvolalo první přílivovou vlnu nadšení z počítačů v Československu. Sinclairisté sdružení v největší pražské „počítačové" svazarmovské základní organizaci se tak rázem stali nejsilnější skupinou, neboť do té doby samostatně působící klub ZX 81 splynul se Spektrem. Teprve dalším rozšiřováním řady počítačů Sinclair spektrum — +, +1, +3 a QL, se opět klub rozpadá na specializované pracovní skupiny i když pod jednotným vedením.

Takto organizovaný působí především v budově Městské stanice techniků v Praze 6, Pod Juliskou 2. Pravidelné schůzky klubu sdružujícího dnes více než patnáct set členů, mají program, v němž se střídá burza softwaru s odbornými přednáškami. Samozřejmě, že tím činnost nekončí. Klub půjčuje zájemcům technickou literaturu, pořádá konzultační dny, účastní se různých veřejných akcí. Ale také, což nás zaujalo, má patronát nad Střední ekonomickou školou v Resslově ulici, kde pomáhá zvyšovat úroveň výuky výpočetní techniky tím, že poskytuje tamním profesorům odborné konzultace a úspěšně rozvíjí zájem studentů o práci s vyspělou a pro většinu mladých lidí přitažlivou technikou.

V této oblasti mají obrovské zkušenosti. Však také péče o své členy-začátečníky patří v tomto klubu k nejhodnotnějším. Do vínku jim poskytují zdarma úplné nejzákladnější softwarové vybavení. Do něj patří vedle databanky klubu i textový procesor, ladící prostředky, popřípadě vedle basicu i další programovací jazyky. Jaké to je bohatství pro začínajícího příznivce není nutné zvlášť komentovat. A když se k tomu přidá další neméně významná pomoc, to je pořádání dvoudenních až sedmidenních seminářů pro začátečníky, pak patrně začátečníci z jiných klubů mohou jen závidět.

Klubové burzy také mají jistou odlišnost od jiných. Sice tak jako všude ve svazarmovských organizacích zabývajících se počítačovou technikou platí i zde nekompromisní pravidlo, že výměny programů neznají finanční spekulace. Jenže na burzy si lidé přinášejí vlastní počítače a datarekordéry, avšak televizory jim zdarma propůjčuje klub. Na jedné straně je tento systém rychlejší v předávání programů na místě, na straně druhé však chtě nechtě vyvolává předhánění se ve využívání televizorů a tím i nervozitu přítomných.

Od dubna 1987 vychází klubový zpravodaj. Má mít čtvrtletní periodicitu. Pokud jím zalistujete naleznete řadu zajímavých informací, odborné články, výpisy zajímavých programů a dokonce i herní návody. Je jen škoda, že má poloviční náklad než čítá členská základna. A ještě jednu poznámku, kterou nám snad soudruzi dávající dohromady časopis odpustí. Při malém nákladu a šestnáctistránkovém rozsahu je velkým přepychem předkládat zájemcům o počítačovou techniku, hladovým především po informacích z této oblasti, začátečnické pokusy o literární žánry na téma počítače, byť by šlo o horory, sci-fi nebo povídky.

Zvláště ostře kontrastuje takový pokus se strohým vyjadřováním jednoho z vedoucích funkcionářů této svazarmovské organizace a dlouholetého předního propagátora klubu soudruha Daniela Dočekala, kterého jsme se zeptali na některé zajímavosti z klubové činnosti:

Kdo u vás zajišťuje rozšiřování programů pro začátečníky?

Banda nadšenců.

Co chcete udělat pro zlepšení situace se stísněnými prostorami na burze?

To se nedá zlepšit, protože prostory nám nikdo nedá a není je kde vzít. Už se o to snažíme asi čtyři roky.

S nárůstem členů pro vás vznikají nové problémy. Jaké?

Že je není kam dát.

V čem je atypie Sinclair klubu?

V masovosti a záběru činnosti vzhledem k tomu kolik má lidí.

Snažíte se spolupracovat s jiným podobným klubem v ČSSR? Jak to jde?

Dost těžko, protože nevíme, kde ty kluby jsou. Neexistuje centrální evidence.

Zajímavosti v klubu?

Lidé zásadně nečtou papíry, které jsou jim předkládány.

Jaký je hit klubu v oblasti programového vybavení?

Nelze říct.

Ale přece se v každém klubu v určité chvíli o něčem více diskutuje?

Nelze říct.

Jaký je například nejoblíbenější program mezi členy v počítačových hrách?

To se značně pohybuje.

Pořádáte třeba soutěže pro děti?

Nepořádáme, protože to je věc nadřízených orgánů. Svazarm je organizace pro dospělé lidi. Pro děti má sloužit SSM. Neslouží a PO SSM už vůbec.

???

Můžete našim čtenářům říct, jak to výhledově vypadá s počítači Sinclair?

Jsou u konce, protože nemají víc co dát. Je to stále variace na starý typ Sinclaira.

Co se tedy stane, pokud firma nepřijde s něčím novým? Klub zanikne?

Klub by každopádně zanikl. Sice za dost dlouho, protože Sinclairů je u nás hodně.

Otázek samozřejmě bylo více, ale protože D. Dočekal, jak řekl, se při nich nudil, raději jsme tento zajímavý rozhovor ukončili. Snad i tak čtenářům ledacos řekne, či spíše napoví.

/MŘ/

1/ Počítačová gramatika

Píše ing. Rudolf Pecinovský, CSc.

PŘEDMLUVA

S novým ročníkem časopisu otevíráte i nový kurs. V minulém roce jste na této stránce nacházeli Slovník výpočetní techniky, který se vás snažil seznámit se základními pojmy, s nimiž se můžete setkat v literatuře o počítačích. Od tohoto čísla otevíráme kurs, v němž vás budeme chtít seznámit se základy programování. Předem však varujeme, že náš přístup bude poněkud netradiční. Autoři naprosté většiny dosavadních učebnic programování si vždy vybrali nějaký konkrétní programovací jazyk a po delším nebo kratším (a nebo také žádném) úvodu, zabývajícím se vývojovými diagramy, se jali učit čtenáře programovat v daném programovacím jazyku.

Uvedený postup má několik nevýhod:

1. Autor musí ve svém výkladu respektovat omezení daná jazykem.
2. Žáci se zejména v počátcích výuky nemohou plně soustředit na vlastní programování jako takové, ale velice často je jejich pozornost odváděna některými detaily jazyka, jako kam patří a kam nepatří středník, kde se píší kulaté a kde hranaté závorky apod.
3. Jazyky vhodné pro výuku mají nejrůznější omezení, která nedovolují přirozeně naprogramovat některé obraty, které se pak musí nějakým složitým a pracným způsobem obcházet. Jazyky, v nichž lze tyto obraty naprogramovat relativně přirozeně, zase nebývají z nejrůznějších příčin vhodné pro výuku.
4. Absolventi takovýchto kursů (myslíme tím úspěšné absolventy) se naučí v daném jazyku programy nejen zapisovat, ale i vymýšlet.

Někteří z vás se možná budou podivovat, proč jsme ke svým námitkám zařadili také čtvrtý bod. Programátoři, kteří „šijí" programy na svůj oblíbený jazyk, nejsou schopni se přizpůsobit v rozumné podobě novému jazyku, byť by byl sebelepší. Okruh problémů, které jsou schopni řešit je dlouho omezen jazykem, který se naučili jako první. Někteří vyučující (např. zakladatel hnutí za moderní programování E. W. Dijkstra) dokonce odmítají do svých kursů moderního programování přijmout studenty, kteří již programovali v jazyce Fortran nebo dokonce v jazyce Basic.

My se budeme v tomto kursu držet metodiky, která se v současné době prosazuje ve všech zájmových kroužcích. Jako základní výrazový prostředek, v němž budeme vymýšlet své programy, zvolíme tzv. **kopenogramy.** Ty mají tu výhodu, že obdobně jako vývojové diagramy nejsou vázány na žádný programovací jazyk. Pokud tedy nevymýšlíme programy v kopenogramech, můžeme pak prakticky libovolně střídat programovací jazyky a výsledný program vždy přepsat do toho jazyka, který bude dané aplikaci a našim možnostem vyhovovat nejlépe. Kopenogramy jsou prostě schopny provázet programátora od jeho prvních krůčků až do jeho programátorského důchodu.

Další naší odlišností od zavedených zvyklostí bude to, že se nebudeme snažit čtenáře naučit vše najednou, jak tomu bývá většinou zvykem. Přidržíme se raději zásad, které hlásal již J. A. Komenský a budeme vás zasvěcovat do jednotlivých tajů programování postupně.

Při tvorbě programu je třeba projít několika základními stádii. V prvé fázi je třeba definovat, co má program všechno umět. Ač se to na prvý pohled nezdá, tato fáze je na programátorovy znalosti a zkušenosti nejnáročnější. Vyžaduje od návrháře největší zkušenosti a nejvíce se podepisuje na kvalitě výsledného díla.

V další fázi se musí definovat, jakým způsobem bude zpracovávaná skutečnost reprezentována v paměti počítače. Navrhují se potřebné datové struktury. Na kvalitě tohoto návrhu také velmi záleží, protože velice výrazně ovlivňuje jak rychlost tvorby celého programu, tak i efektivitu výsledného díla.

Teprve nyní nastupuje to, co si většina lidí představuje pod programováním: vlastní návrh algoritmů, tj. postupů, kterými bude počítač plnit zadané úkoly. A protože je tato fáze nejjednodušší, začneme právě s ní.

1. **KAREL**

Výuka programování, ostatně jako každá jiná výuka, ztrácí bez zpětné vazby velice rychle na efektivnosti. Učit proto programování a nemít k dispozici počítač je přinejmenším velmi odvážné. Jak jinak totiž mohou žáci poznat, že jimi vytvořené programy jsou správné a že veškerou látku dobře pochopili.

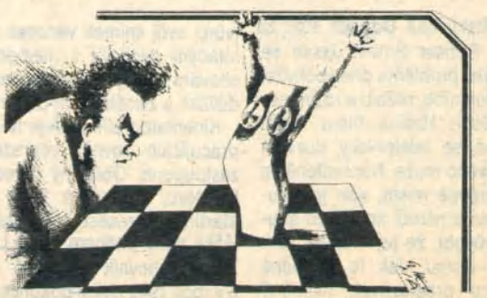
Kromě toho jsme na počátku vyhlásili, že se vás budeme snažit nejprve naučit algoritmizovat a teprve poté si začneme vyprávět o datech. Teorie algoritmů však tvrdí, že algoritmus, který nezpracovává žádná data je čirý nesmysl. Uděláme však malý podfuk — zavedeme si taková prostředí, v němž naše programy nebudou používat data zjevně, ale skrytě. Tímto prostředím bude svět robota Karla.

Mnozí se budou ptát, proč jsme si vybrali na pomoc právě tuto postavičku. Proč? Protože jsme zatím v celém světě nenašli žádný lepší prostředek pro výuku algoritmizace. Karel má totiž oproti klasickým metodám výuky několik nesporných výhod.

- Celý systém je maximálně jednoduchý. Není na něm prakticky nic k učení, takže jej pochopí i šestileté děti. Žáci se proto mohou při výuce plně soustředit na přednášenou látku.
- Programy pro Karla jsou nesmírně názorné, takže žáci velice rychle pochopí vnitřní zákonitosti a smysl jednotlivých konstrukcí a obratů. Mohou si prostě „osahat" jak co funguje.
- Systém nemá žádné datové struktury a proto je učitelé (a s nimi i žáci, kteří se seznámili s programováním již dříve) nemohou používat. Jsou proto nuceni řešit všechny úlohy pouze vhodným algoritmem, což opět vede k rychlejšímu a hlubšímu zvládnutí látky.
- Na rozdíl od klasických výukových příkladů typu „najdi největšího společného dělitele" připadá výuka za pomoci Karla žákům od samého počátku zajímavá. Můžete sice namítnout, že se líbí dětem, nemusí se ještě líbit dospělým, ale doufáme, že příklady pro Karla vás zaujmou přinejmenším stejně, jako největší společný dělitel.

Zanechme však již přesvědčování o správnosti nastoupené cesty a podívejme se na Karlovy vlastnosti. Karlovým životním prostorem je dvorek, který si můžeme představit rozdělený na čtverce stejně jako např. šachovnice. Karel může být natočen do jedné ze čtyř světových stran a na povel se přesunout na sousední políčko ve směru, do nějž je natočen. Kromě toho je schopen na políčko, na němž stojí, pokládat značky a nebo je z něj naopak zvedat.

Kromě výše zmíněných akcí je Karel „schopen" provádět i některé testy. Dokáže zjistit, zda je před ním zeď, zda je pod ním značka, zda je natočen do dané světové strany a na některých počítačích i to, zda někdo stiskl klávesnici. Tím výčet jeho schopností končí. Vše ostatní jej musíme naučit sami. (pokračování)



BURZA PROGRAMŮ

3× BASIC pro ATARI 800 XL

Zkoušeli jste si již udělat hlavičku pro svůj program? Jestliže ano, víte, že to není nic jednoduchého, pokud chceme, aby byla hezká a netradiční. A právě jeden takový návod vám předkládáme. Na obrazovce se střídají barevné pruhy, které „plují" shora dolů. Do nich je možné velice jednoduše vpisovat různé nápisy. Vyzkoušejte také vynechat příkaz GRAPHICS a uvidíte, co ještě program dokáže.

```
1 REM Rolovani barev — ATARI 800 XL
2 GRAPHICS 2+16:POKE 708,14
3 POSITION 6,4:? #6;"BIT KLUB":POSITIO
N 8,6:? #6;"—SP—"
4 FOR A=0 TO 63:READ APC:POKE 1664+A,A
PC:NEXT A:POKE 77,0:A=USR(1664):END
5 DATA 173,36,2,141,193,6,173,37,2,141
,194,6,160,180,162,6,169,7,32,92,228,2
38,192,6,173,192,6,141,10,212,141
6 DATA 26,208,166,77,224,3,208,238,172
,193,6,174,194,6,169,7,32,92,228,104,9
6,206,200,2,173,200,2,141,192,6
7 DATA 76,98,228
```

Co všechno umí tento kratičký program? Po zadání příkazu RUN se počítač ptá: Pocinaje radkou . . .? Sem vložíte číslo řádky od které chcete DATA vkládat. Jeho další otázka: Po kolika . . .? vám umožňuje rozhodnout po kolika budou řádky odstupňovány.

```
1 DIM A$(150):?"↖":? „Pocinaje radkou
. . . ";:INPUT LINE:? :? „Po kolika . . . . .
. . . ";:INPUT PO: ? "↖"
2 INPUT A$:GRAPHICS 0:POSITION 2,4:? L
INE:;"DATA";A$:? „CONT":POSITION 2,0:PO
KE 842,13:STOP
3 POKE 842,12:LINE=LINE+PO:GOTO 2
```

Malou úpravou dosáhnete toho, že program bude fungovat jako automatický číslovač řádků. Tato úprava spočívá:

```
2 INPUT A$:GRAPHICS 0:POSITION 2,4:? L
INE;A$:? „CONT":POSITION 2,0:POKE 842,
13:STOP
```

V obdobné verzi lze použít programu pro vymazání většího množství nepotřebných řádek:

```
1 ? "↖":? „Pocinaje radkou . . .";:INPUT
LINE:? :? „Po kolika . . .  .";:INPUT
PO:?
2 ? „Do radky . . .";:INPUT DO:?
" ↖"
3 GRAPHICS 0:POSITION 2,4:? LINE; "":?
„CONT":POSITION 2,0:POKE 842,13:STOP
4 POKE 842,12:IF LINE=DO THEN 6
5 LINE=LINE+PO:GOTO 3
6 ? :? „Skonceno !":END
```

Chcete vědět na jaký den připadá vaše narození? Nebo v který den porazil Žižka křižáky na Vítkově? Byla to sobota, pátek či pondělí? Na tyto otázky vám pomůže odpovědět náš program.

```
1 ? "↖":SETCOLOR 2,0,0:? „Vloz rok, mes
ic, den . . .";:INPUT R,M,D:IF M<3 THEN R=
R-1:M=M+12
2 D=R+INT(R/4)-INT(R/100)+INT(R/400)+3
*M-INT(M+M+1)/5)+D+1:D=D-INT(D/7)*7
3 IF D=0 THEN ? „Nedele"
4 IF D=1 THEN ? „Pondeli"
5 IF D=2 THEN ? „Utery"
6 IF D=3 THEN ? „Streda"
7 IF D=4 THEN ? „Ctvrtek"
8 IF D=5 THEN ? „Patek"
9 IF D=6 THEN ? „Sobota"
```

(MŘ)

Vážení čtenáři,
chceme vám připomenout, že na této stránce chceme otiskovat nejen seriál a různé informace pro vaši potřebu, ale že očekáváme i vaši aktivní spolupráci. Rádi otiskneme zprávy z činnosti klubů, zkušenosti z vaší praxe s různými počítači, zodpovíme vaše dotazy a pro potřeby vašich kolegů otiskneme i vámi zaslané krátké programy, podobné těm, které pro toto číslo připravil náš spolupracovník . . . . . Nezapomeňte, že tato stránka je určena vám, majitelům osobních počítačů, a že i na vás záleží, jaká bude její tvář, její užitečnost.

Jako první vlaštovku otiskujeme informaci z dopisu tajemníka 602. ZO Svazarmu Josefa Kroupy:

„602. ZO Svazarmu ve spolupráci s Obvodním domem pionýrů a mládeže Praha 1 rozšiřuje svoji činnost o Klub počítačů Commodore Amiga. Schůzky se konají pravidelně každé úterý od 17.00 do 19.00 hodin v budově ODPM Praha 1, Školská 15. Přijďte osobně nebo projevte svůj zájem korespondenčním lístkem na adresu: 602. ZO Svazarmu, Wintrova 8, 160 41 Praha 6."

Vaše příspěvky očekáváme na adrese: Redakce Svět práce, Václavské nám. 17, 112 58 Praha 1. Obálku označte heslem „bit klub SP". Příspěvky, které otiskneme, budeme též honorovat. **Redakce**

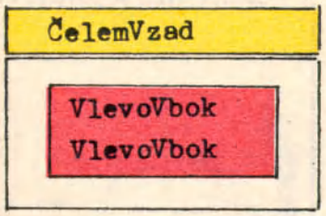
Počítačová gramatika

2/ Píše Ing. Rudolf Pecinovský, CSc.

2. PROCEDURA — ZÁKLADNÍ PRVEK PROGRAMU

V minulé kapitole jsme si vysvětlili, že Karel prakticky nic neumí. Budeme-li tedy po něm chtít jakoukoliv složitější činnost, budeme jí muset nejprve Karla naučit. V této souvislosti budeme používat dva termíny. Příkazy (a později i podmínky), které Karel zná od počátku, budeme nazývat **primitiva;** příkazy, které jej naučíme, budeme nazývat **příkazy odvozené.** Těm z vás, kteří již o programování něco vědí, prozradím, že primitiva vlastně odpovídají strojovým instrukcím, resp. klíčovým slovům daného programovacího jazyka, a slova odvozená jsou ekvivalenty našich programů a podprogramů.

Víme tedy, že Karel zná čtyři primitiva. Krok, VlevoVbok, Polož a Zvedni. Začněme tedy s vlastním učením a ukažme si, jak bychom mu mohli definovat nějaký odvozený příkaz.



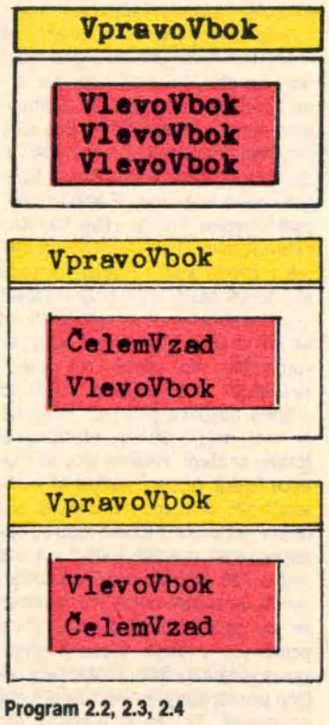
Program 2.1

K jedněm z nejjednodušších patří příkaz ČelemVzad. Jeho definice by mohla vypadat následovně:

Na tomto nejjednodušším příkladě si ukážeme a vysvětlíme některé nezákladnější konvence týkající se kopenogramů.

1. Všechny programy zapisujeme do obdélníků, které nazýváme bloky.
2. Název programu (podprogramu, procedury . . .) píšeme do horního záhlaví a od zbytku programu jej oddělujeme dvojitou čarou. Záhlaví s názvem programu (a obecně s jakýmkoliv názvem) vybarvujeme žlutě.
3. Posloupnost akcí, které má počítač vykonat, zapisujeme do bloků, které vybarvujeme červeně.
4. Nemáte-li možnost použít barvy, je vhodné vybarvované bloky odlišit alespoň šrafováním.

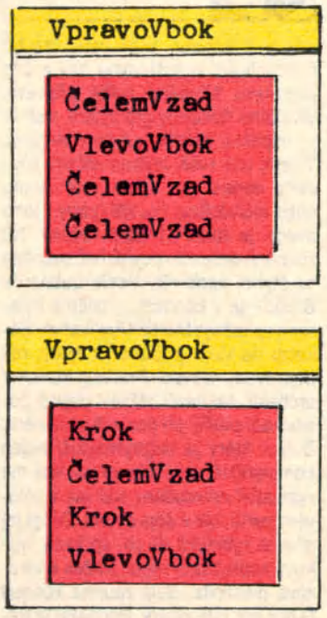
Základní pravidla tedy známe. Zkusme proto nyní naučit Karla třeba příkaz VpravoVbok. Zamyslíte-li se nad problémem podrobněji, lehko přijdete na to, že k uvedenému cíli se můžete dostat několika ekvivalentními způsoby:



Program 2.2, 2.3, 2.4

V prvé definici jsme použili pouze primitiva VlevoVbok, v druhé a třetí jsme využili toho, že Karel již zná slovo ČelemVzad. Všechny tři definice vedou ve svém důsledku k správné činnosti a proto je musíme považovat všechny za správné.

Existuje samozřejmě nekonečné množství dalších definic, ale Karel při jejich plnění již vykonává některé zbytečné činnosti, např.:



Program 2.5, 2.6

I takovéto definice musíme pokládat za správné, pokud podle nich počítač splní žádaný úkol. Podíváme-li se na kteroukoli z prvých čtyř definic slova VpravoVbok (Programy 2.2 až 2.5), vidíme, že Karel při jejich plnění zůstává na místě a proto je bude vykonávat bezchybně kdekoliv.

Vyzkoušejte si ale poslední definici (Program 2.6) v případě, kdy Karel stojí čelem u zdi. Dostane-li nyní příkaz VpravoVbok, bude muset udělat podle definice nejprve krok, ale tento krok je krokem do zdi! Karel si, obrazně řečeno, rozbije nos. Přitom však není pádného důvodu pro to, aby Karel neuměl udělat vpravo vbok i v tomto případě.

Definici, která nevyhoví za jakýchkoliv okolností, nemůžeme považovat za správnou!

Z uvedeného programu pro nás plynou dvě poučení:

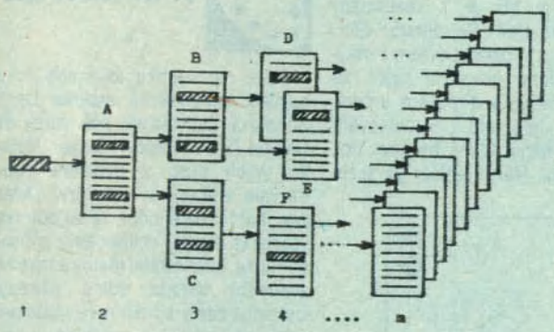
1. Není tak důležité, jak je ta která činnost definována, tj. jakým postupem se dosáhne cíle. Důležité je to, zda při plnění tohoto programu počítač za jakýchkoli okolností splní zadanou úlohu.
2. Chyby v definici nemusí být na první pohled patrné. Proto když vytvoříme nějaký program, a to nejen celý program ale i jakoukoliv jeho smysluplnou část, musíme se ihned přesvědčit, zda jsme jej nadefinovali správně.

STRUKTURALIZACE VÝVOJOVÝCH DIAGRAMŮ

Strukturalizace je vlastně zásada jednoduchosti řídících struktur, tj. jednoduchost prostředků, jimiž je definováno pořadí realizace příkazů vyjádřených vývojovým diagramem. Strukturované programování se tedy opírá o respektování určitého počtu zásad, z nichž zásadní jsou:

a) **princip abstrakce** — představuje vlastně základní přístup k tvorbě programu. Vede programátora k promyšlenému členění programu na dílčí funkční části, které nejprve vymezí hrubě a v dalších krocích je konkretizuje;

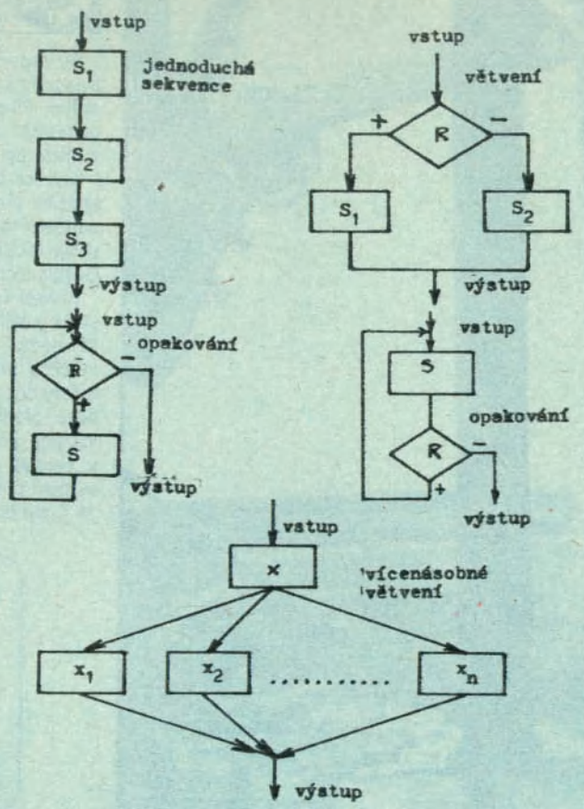
b) **programování shora dolů** — znamená konkrétní aplikaci abstrakce. Začíná se popisem úlohy a jejího řešení. Dalšími příkazy se dosahuje stále většího zjemnění, až nakonec dosáhneme úrovně zvoleného programovacího jazyka. Celý postup znázorňuje **obr. 1.**;



Obr. 1. Programování shora dolů
A, B, C, D, E, F — programovací formuláře
1, 2, 3, 4, ... m — úrovně postupné konkretizace

c) **zásada striktního používání základních řídících struktur** — vede programátora k tomu, aby při sestavování programu používal pouze tří zvolených elementárních řídících struktur, které postačují k tomu, aby vyjádřil přehledně procedurální část programu. Mimo to se dovoluje používat i vícenásobné větvení programu **(obr. 2);**

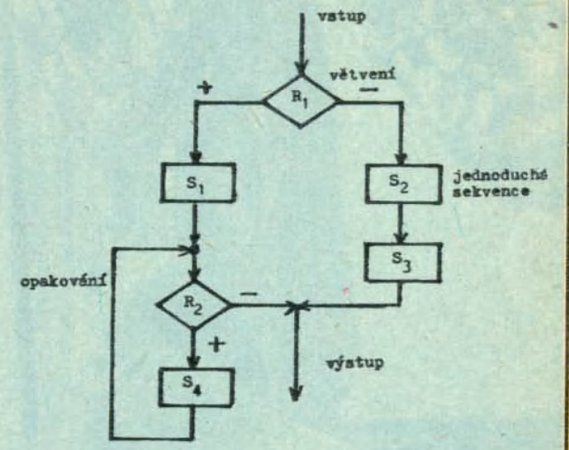
d) **zásada jediného vstupu a jediného výstupu** — tuto zásadu se doporučuje striktně dodržovat při spojování základních řídících struktur, při konstrukci složitějších řídících struktur a jejich



Obr. 2. Základní řídící struktury

sestavování do zdrojového programu tak, aby plnil všechny funkce, které požadujeme **(obr. 3);**

e) **zásada programování bez skokových příkazů** — chceme se její pomocí vyhnout používání příkazu GO TO, což podstatně napomáhá zvýšit přehlednost a srozumitelnost programů. Jde o poměrně radikální zásadu; program GO TO je dosti často používán. V rámci strukturovaného programování se má



Obr. 3. Složená řídící struktura

k tomuto účelu využívat základní řídící struktury pro větvení. Někdy se připouští pouze omezené používání příkazu GO TO, a to zpravidla ve vztahu k použitému počítači (omezené řídící struktury);

f) **zásada omezení velikosti jednotlivých programových celků** (modulů, segmentů, procedur, sekcí, podprogramů ap.). Touto zásadou se snažíme rozdělit program na přehledné části, přičemž každá programová jednotka je logicky uzavřená a samostatná část programu a vyjadřuje, formou naprogramovaného logaritmu, jistou dílčí funkci, která může být vnímána jako celek ve své úplnosti.

Názory na velikost programových jednotek se však různí — skutečná velikost však nemůže být rozhodující, ale musí být pouze prostředkem k dosažení přehlednosti a zvladatelnosti programu.

Strukturované programování se dnes všeobecně považuje za největší pokrok v oblasti metodiky programování, poněvadž poskytuje řadu výhod.

Ing. IVAN SENJUK

Počítačová gramatika

3/ Píše Ing. Rudolf Pecinovský, CSc.

Ukažme si nyní jiný příklad. Dejme tomu, že bychom chtěli, aby náš Karel uměl skákat jako šachový kůň, tedy do „L". Jak mu tento příkaz vysvětlit? Jedna z možností je:

```
JakoKůň
    Krok
    Krok
    VpravoVbok
    Krok
```

Opět to není možnost jediná. Ale tentokrát je situace složitější. Žádaný pohyb totiž můžeme nejen nadefinovat několika způsoby, ale v zadání není ani jasně řečeno, který z řady možných tahů koně je ten pravý a není tam ani uvedeno, do kterého směru má být Karel po splnění úkolu natočen.

Tato nejednoznačnost zadání nemusí být vždy chybou. Někdy může být naopak záměrná. Pokud totiž na některých detailech nezáleží, lze ponechat na programátorovi, aby zvolil takové řešení, které mu bude nejvíce vyhovovat, nebo které se mu bude nejsnadněji programovat.

Bude-li se vám proto někdy zdát, že zadanou úlohu můžete řešit několika způsoby, vyberte si ten, který vede k řešení nejrychleji.

Myslím, že vysvětlování bylo již dost, a že je nejvyšší čas si nabyté vědomosti nějak prověřit. Pokuste se proto sami definovat příkazy: Dvojkrok, Zpět (Karel udělá krok „pozpátku"), ÚkrokVlevo, ÚkrokVpravo.

3. SLOŽITĚJŠÍ ÚLOHY — DEKOMPOZICE

Jak si jistě sami domyslíte, většina úloh, které má počítač řešit, nevede na tak jednoduché programy, s jakými jsme si „hráli" v minulé kapitole. Naopak. V praxi se velice často setkáme s programy, v nichž je jednotlivých příkazů několik desítek či stovek tisíc. A ty již nejde řešit „z voleje". K jejich úspěšnému řešení je naopak nutná velice důkladná analýza celého problému.

Základním strategickým heslem je staré římské „Rozděl a panuj!" Cílem veškerého snažení tedy je rozdělit celý problém na rozumně velkou množinu rozumně malých podúloh, a to tak, abychom mohli dále řešit jednotlivé podúlohy pokud možno samostatně.

Pamatujte si:

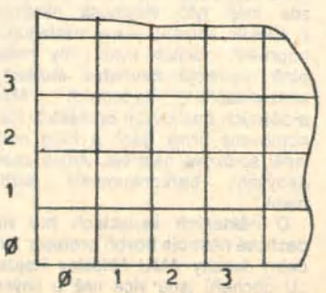
Čím později usednete k počítači, tím dříve budete s programem hotovi.

Na první pohled vypadá tato zásada paradoxně, ale věřte, že je potvrzena dlouholetou zkušeností řady programátorů. Dostat úkol a ihned sednout ke klávesnici, toho je schopen pouze zarytý začátečník. Veškerý čas, který zpočátku věnujete důkladné analýze úlohy a promyšlení celkového řešení se vám později mnohonásobně vrátí jako čas ušetřený při opravování a upravování nedomyšlených vzájemných vazeb a opravách nejrůznějších chyb.

Nepodlehněte proto počátečnímu zdání, že problém je naprosto triviální, a věnujte vždy alespoň chvilku jeho rozmyšlení nad papírem. Neuspěchávejte zasednutí za klávesnici počítače. Začněte jej „fukat" do počítače až tehdy, máte-li ve všem jasno.

Dosti již ale mentorování a ukažme si celý postup opět na příkladech. Pověřme např. Karla úkolem vyznačkovat na dvorku čtverec o straně dlouhé tři políčka **(obr. 3.1.)**. Většina

Obr. 3.1



začátečníků by po obdržení takovéhoto zadání zasedla ke klávesnici a napsala by přibližně následující program:

```
Čtverec 3A
    Polož
    Krok
    Polož
    Krok
    Polož
    VlevoVbok
    Krok
    Polož
    Krok
    Polož
    VlevoVbok
    Krok
    Polož
    Krok
    Polož
    VlevoVbok
    Krok
    Polož
```

Program 3.1 Čtverec 3A

Počítačové zajímavosti

Programová nabídka v NSR

V NSR se zabývali rozborem prodeje programového vybavení pro malé počítače podle aplikací. Největší podíl mají elektronické hry s 61 %, dále jsou to organizační programy s 15 %, systémy se 14 % a vzdělávací programy s 10 %. Zvýšení zájmu o programy pro hry v NSR se vykládá platností nového zákona na ochranu mládeže, který mladým do 18 let zakazuje přístup do veřejných heren. Celkový počet evidovaných programů převýšil již 5 tisíc, z toho 62 % připadá na systémy tří výrobců: Commodore, Schneider a Atari.

Největší výrobci polovodičů

V roce 1986 došlo poprvé v historii k překvapení: v pořadí největších výrobců polovodičových součástek se na prvních třech místech neobjevila žádná americká firma. Nyní je pořadí prvních pěti společností: 1. NEC, 2. Hitachi, 3. Toshiba (všechny Japonsko), 4. Motorola, 5. Texas Instruments (USA).

Projekční zařízení pro počítače

Firma KODAK vyvinula a prodává systém Datashow, který umožňuje promítnout výstup z počítače přímo na projekční plochu. Princip spočívá v prosvěcování transparentního stínítka s kapalnými krystaly připojeného na počítač. Vzniklý obraz se promítá na plátno nebo stěnu. Zařízení lze připojit ke každému malému či osobnímu počítači a vejde se do kufříku. Využití je především při konferencích, školeních, předvádění programů apod. Počítač umožňuje měnit zvětšení obrazu, vybírat detaily, kombinovat obrazy. Zařízení stojí cca 3600 švýcarských franků.

Ochrana japonských programů

V Japonsku bylo zřízeno centrum, které registruje a bude 50 let archivovat všechny programy vytvořené v Japonsku. Po tuto dobu bude centrum chránit autorská práva tvůrců programů. Tento systém má napomáhat prosazování domácích programů a rovněž prověřuje, zda dovážené programy nejsou stejné nebo podobné domácím programům.

Strach z počítačů

V NSR byla provedena studie o trendech v mikropočítačové technice. Podle ní 78 % západoněmeckých občanů je názoru, že konkurenceschopnost hospodářství NSR bezpodmínečně vyžaduje nasazení počítačů. Strach z nezaměstnanosti, která je spojena s využíváním počítačů, má 10 % dotázaných (rychle se snižuje). Obavy z „komputerizované" společnosti má 58 % těch, jejichž pracoviště má být v blízké budoucnosti vybaveno osobním počítačem a pouze 38 % těch, kteří již na pracovišti osobní počítač využívají.

(FI)

Počítač áno a čo ďalej?

Dnes už skoro každý vie, alebo si aspoň niečo myslí o tom, čo nazývame počítač. Preto si dovolím ponúknuť svoje úvahy bez toho, že by som ho presne definoval. Len úplne abstraktne môžeme povedať, že jeho úlohou je poskytnúť informáciu vždy, keď ju potrebujeme. Pokúsim sa stručne rozobrať jednu stránku ich nasadenia a to, ako sa vlastne využívajú a aký je ich prínos pre spoločnosť. Nekladiem si za úlohu vystihnúť túto problematiku v celej šírke, ale som presvedčený, že táto téma si zaslúži širšiu diskusiu. Počítače sa líšia tak svojimi možnosťami, nadobúdacou cenou, ako aj udržiavacou cenou, to znamená nákladmi na personál a náhradné diely, **ktoré zabezpečujú jeho prevádzky**schopnosť. Vzťah medzi jeho cenou a možnosťami ako aj inými nákladmi rozhodne nie je vždy priamoúmerný.

Je určené, že počítač musí pracovať na dve smeny. Zdá sa, že ak to dodržiavame, je všetko v poriadku. Čiže musíme ho zapnúť so všetkými periférami, nahnať, alebo prilákať k tomu ľudí, aby ho začali kŕmiť, alebo pokračovali v kŕmení s údajmi. Čiže všetko je v poriadku, počítač beží a využíva sa. No a tu sa treba zastaviť.

Keď ponúkam strojový čas, tak si zaň vyberám náhradu. **Je mi celkom** jedno čo si tam kto robí, samozrejme v dovolenom rozsahu a keď on má veľa peňazí a je nevyťažený počítač, tak čím dlhšie robí riešenie, tým dlhšie mám ja bezstarostný život. Treba tu ale podotknúť, že to platí len ako uvádzam, pretože stav domácej výpočtovej techniky, jej spoľahlivosť, respektíve skôr nespoľahlivosť vyžaduje enormné úsilie od vlastníka na to, aby ho udržal v prevádzke.

Zoberme si príklad, že robíme spracovanie. Určité rutinné spracovanie trvá povedzme päť hodín. Všetky rutiny nám denne vykryjú celú nutnú prevádzku. Trápi nás znova niečo? Nič. To trvá dovtedy, pokiaľ sa neobjaví nedostatok strojového času. Začne sa redukovať, špekulovať, ako z toho von. Stále sa vám zdá, že je všetko v poriadku? Zaujímalo niekoho, alebo bol niekto ekonomicky motivovaný k tomu, aby povedzme rutinné spracovanie netrvalo päť, ale len tri hodiny? Zaujíma to vždy niekoho aj dnes? Čiže nevhodne postavené spracovanie so zbytočnými operáciami a vstupnými údajmi zatiaľ veľa razy napĺňa výkony organizácií, ale vo svojej podstate okráda spoločnosť minimálne o zbytočne spálenú energiu. Keď k tomu prirátame spotrebu pracovnej sily, tak je to aj mrhanie ľudskou prácou. Je vinovaté to nariadenie, ktoré určuje, koľko má počítač pracovať a aký má byť z neho prínos?

Iste, nariadenia sa nevymýšľajú samoúčelne a je úplne jasné, že mať počítač ako ozdobu je nezmysel. Veľakrát si však neuvedomujeme, nechceme, alebo nie sme schopní pochopiť, že vlastne prácu predstierame. A okrem morálnych dôvodov na zmenu daného stavu, a aj to len sporadicky, zatiaľ iné nie sú. Nechcem tým tvrdiť, že sme celkom márnotratná a bezduchá spoločnosť, ale žiaľ, niektoré veci sme si zvykli nevidieť a čo horšie, naučili sme sa všetko vysvetliť. K tomu nám slúži množstvo nariadení, smerníc a podobných umŕtvujúcich papierov. Veď aký je priepastný rozdiel medzi spoločensky nutnou a skutočne odvedenou prácou.

Zbaviť sa nákladového hľadiska, že koľko napríklad rutinné spracovanie trvá a ponechať iba cenu spoločensky žiadúcu, to znamená, koľko je to odberateľovi hodné, samozrejme v rámci dopytu a ponuky, ihneď sa kvalitatívne hľadisko stane prvoradým, bez toho že by sa centrálne určilo, koľko má byť počítač denne zapnutý, a koľko za každú činnosť, až do kývnutia prstom, sa má odberateľovi vyfakturovať.

Vlastník počítača by sa v tom prípade dôsledne staral o to, aby všetky rutiny trvali čo najkratšie, aby mohol uspokojiť aj ďalších záujemcov. Získal by tým nielen osobný prospech, ale súčasne by aj viac odovzdal spoločnosti a to by ani nespálil viac energie.

Formu ponúkania spracovania na počítači, napríklad miezd, som si ako všeobecne rozšírenú čiastočne vymyslel. Je to možno moja chyba, ale neviem o takej organizácii, kde by som prišiel, povedal čo chcem, oni mi povedia aké vstupné údaje potrebujem a potom mi zabezpečia spracovanie za rozumnú cenu. Stretávam sa s tým iba sporadicky. Môže byť takáto služba námetom prestavby?

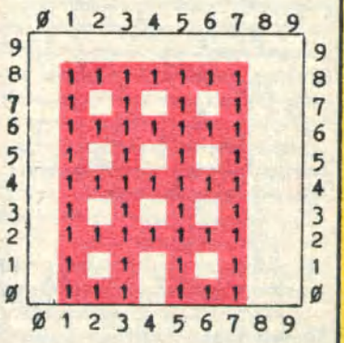
Ing. VLADIMÍR LABÁTH
Ilustrace Vladimír Jiránek

Počítačová gramatika

5) Píše Ing. Rudolf Pecinovský, CSc.

Umění dekompozice složitých úloh na posloupnost úloh jednodušších je jedním ze základních pilířů umění programovat. S jistým zjednodušením lze dokonce říci, že kvalita programátora se pozná podle toho, jak dobře dokáže dekomponovat problém. A protože je tato schopnost a dovednost natolik důležitá, budeme jí ještě chvíli věnovat.

Zkusme Karlovi například vysvětlit, jak „postavit panelák" podle **obr. 3.2**.



Obr. 3.2 Panelový dům

Dříve, než se pustíte do čtení vzorového postupu, pokuste se nejprve navrhnout program pro Karla sami a poté porovnejte vzorové řešení se svým. Neporovnávejte však přesný algoritmus řešení, ale např. celkový počet a celkovou délku procedur řešení svého a našeho.

Tak co, hotovo? Podívejme se tedy společně na jedno z možných řešení našeho problému. Při pohledu na obrázek si většina z vás jistě povšimne, že jednotlivá patra našeho hypotetického domu jsou si podobná jako vejce vejci. Vzápětí by vás proto mělo napadnout, že celý dům lze postavit např. podle **programu 3.5.**

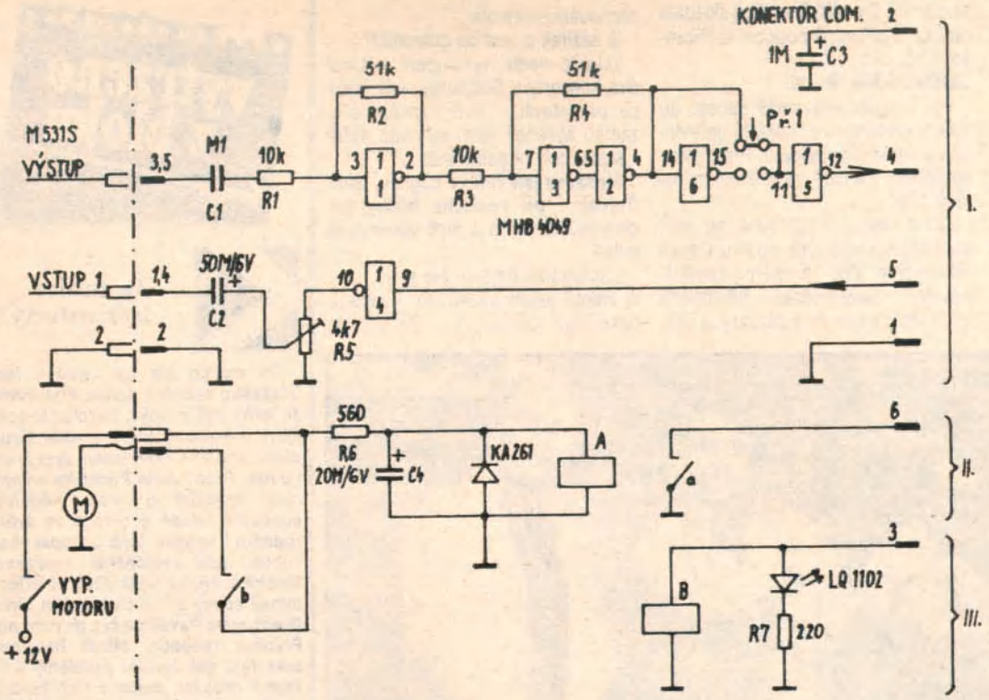
```
Dům
   Základ
   Patro
   Patro
   Patro
   Patro
```

Program 3.5 Dům

STYKOVÝ OBVOD

Pro ukládání programů a dat se u počítačů COMMODORE C116, C16, PLUS 4, C64 kromě diskety používá digitální kazetový magnetofon DATASSETTE. Při použití běžného kazetového nebo cívkového magnetofonu pro počítače COMMODORE je nutný stykový obvod, tzv. INTERFACE. Tento obvod lze vyrobit i amatérsky.

Jedno z možných zapojení, viz schéma, je složeno ze tří funkčních částí. Základem celého stykového obvodu je integrovaný obvod **MHB4049,** který obsahuje 6 hradel **NAND.** V podstatě slouží jako převodník analogového signálu z výstupu magnetofonu na digitální signál úrovně TTL. Ve směru načítání z magnetofonu do počítače jsou zapojeny 4, resp. 5 hradel tohoto obvodu. Ve směru ukládání dat z počítače na magnetofon je použito zbývající hradlo. Odporový trimr **R5** slouží k nastavení správného vybuzení magnetofonu při nahrávání. Přepínačem **př. 1** je možné vynechat jedno hradlo ve směru načítání do magnetofonu. To slouží k nastavení správné „polarity" signálu, aby bylo možné přehrávat i programy uložené originálním DATASSETTEM. Stačí podle kazety s originálním programem nastavit přepínač do polohy, kdy nám počítač správně čte programy. Takto zapojený obvod umožňuje spolupráci počítače s běžným magnetofonem. Jen je třeba spojit špičku **6** (SENSE) se špičkou **1** (ZEM) na konektoru počítače.



Další části stykového obvodu zvyšují komfort obsluhy a nejsou pro funkci nezbytné. Uzemněním špičky **6** dává magnetofon počítači informaci o tom, že magnetofon byl spuštěn tlačítkem PLAY, popřípadě PLAY RECORD, na originálním datarekorderu. Tuto funkci můžeme nahradit relátkem **A,** které je zapojeno do obvodu motoru a při spuštění magnetofonu sepne kontaktem **a** špičky **6** a **1** konektoru COMMODORE.

Třetí část obvodu slouží k zastavení magnetofonu po načtení programu do počítače. Zde použijeme relé **B,** které je ovládáno ze špičky **3** počítače a svým kontaktem přeruší napájení motoru magnetofonu. Svítivá dioda signalizuje chod magnetofonu.

Zapojení obou těchto částí je závislé na použitém magnetofonu a slouží k oddělení napájecích obvodů motoru magnetofonu od elektroniky počítače. Je proto tato část schéma uvedena jako příklad z možných zapojení. V tomto případě byl použit polský stolní kazetový magnetofon M531S. Je třeba zdůraznit, že čím bude použitý magnetofon kvalitnější, bude méně starostí s nahráváním, zvláště u zrychlených (turbovaných) záznamů. Stejně tak to platí o použitých kazetách. Doporučuji kazety s kvalitním páskem a raději kratší, nejvýše C60. Slabší pásek v kazetách C90 je choulostivější, snadno se zvlní a digitální záznam nesnese sebemenší chybu. Při použití stykového obvodu bez ovládacích relé je tedy nutno propojit špičky **6** a **1** konektoru počítače a spouštět magnetofon dřív než odešlete příkaz LOAD, resp. SAVE, příkazem RETURN. Na obrazovce se v tomto případě neobjeví text PRESS PLAY ON TAPE, resp. PRESS PLAY AND RECORD ON TAPE, ale pouze problikne SEARCHING. I zastavení magnetofonu je třeba ovládat ručně.

Stykový obvod vestavíme do vhodné krabičky, např. od rámečků na diapozitivy, na destičku z kuprextitu. Obrazec pro plošný spoj neuvádím vzhledem k obtížné dostupnosti relé. Použité součástky jsou vepsány do schéma a jejich výběr není kritický. Relé jsou na napětí 5 V pokud možno s malým odběrem. Je třeba upozornit na to, že integrovaný obvod MHB4049 je vyroben technologií CMOS, je tedy choulostivý na statickou elektřinu a budeme s ním podle toho zacházet. Znamená to přechovávat ho v antistatickém obalu (alobalu apod.), nedotýkat se vývodů obvodu a pájet uzemněnou páječkou.

ING. MIROSLAV BOHÁČ

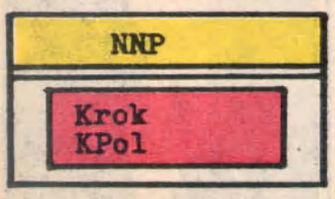
Vážení čtenáři,
omlouváme se za chybu, která se vloudila do programu ve Světě práce č. 2 na této straně. V pátém řádku shora posledního programu vypadla závorka a výraz v tomto řádku má vypadat takto:
*M-INT((M+M+1)/5)+D+1:D=D-INT(D/7)*7

Redakce

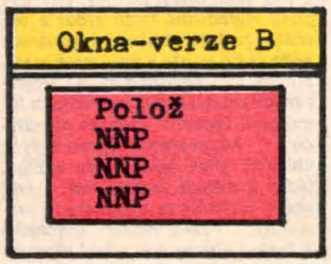
Počítačová gramatika

7) Píše Ing. Rudolf Pecinovský, CSc.

Program 3.14 Okna — verze B



Program 3.15 PNN



Program 3.16 NNP

Zbývá nám ještě dotáhnout do konce proceduru **Strop.** Po chvilce přemýšlení byste asi také přišli na to, že máme štěstí, protože tuto proceduru můžeme naprogramovat celkem jednoduše způsobem ukázaným v **programu 3.17.**

Pokusme se nyní na závěr shrnout základní myšlenky této kapitoly.

1. Nepouštějte se zbrkle do vlastního programování a nejprve se zamyslete nad tím, zda by nebylo vhodné celou úlohu rozdělit na několik jednodušších podúloh.

2. Pokud je ze zadání zřejmé, že se bude některá část programu opakovat, bývá většinou výhodné naprogramovat tuto část jako samostatnou proceduru.

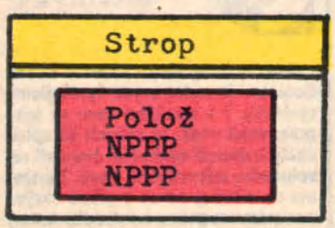
3. Úlohu je v zájmu zvýšení přehlednosti často vhodné rozdělit na podúlohy i v případě, že v ní žádné opakující se části nejsou — viz procedura Patro.

4. Jednotlivé části, na něž úlohu rozdělujete, volte tak, aby tvořily logicky relativně samostatné a ucelené jednotky.

5. Nezapomeňte nadefinovat a zkontrolovat mezimodulové rozhraní. Bez ujasněného mezimodulového rozhraní nepřistupujte k řešení jednotlivých podúloh.

6. Pokud je již dekompozice problému hotova a máte navržené a zkontrolované mezimodulové rozhraní, podívejte se, jestli některé z podúloh již nejsou natolik jednoduché, že je pro vás snadné je přímo naprogramovat. Pokud ano, naprogramujte je.

7. Ze zbylých podúloh vybírejte jednu po druhé a aplikujte na její řešení tyto body. Pořadí řešení není důležité. Někdo vybírá podúlohy od nejjednodušší k nejsložitější, jiný dodržuje pořadí jejich výskytu v programu, třetí volí vlastní, zcela specifický klíč výběru.

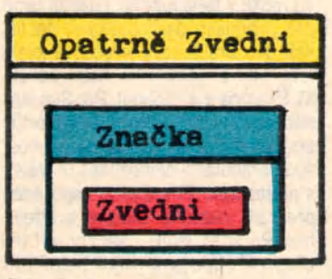


Program 3.17 Strop

4. ROZHODOVÁNÍ V PROGRAMECH

Kdyby počítač opravdu neuměl nic jiného, než vyplňovat prosté příkazy jeden za druhým, nestal by se nikdy takovým všudypřítomným pomocníkem, jakým dnes je. Základní předpoklad inteligentního jednání je umění správně postupovat v závislosti na nastalých podmínkách. Pokud bychom počítač nenaučili volit správnou variantu řešení se zřetelem na nastalou situaci, asi by nám při řešení našich problémů příliš nepomohl.

Dnes si proto povíme o tom, jak naučíme Karla přemýšlet a na základě zjištěných zkušeností se pak rozhodovat. Představte si následující problém: Karel má za úkol zvednout značku. Pokud by však pod ním žádná značka nebyla, mohlo by dojít k chybě. Upravíme proto zadání tak, že Karel má zvednout značku pouze v případě, že pod ním vůbec nějaká značka je. Nadefinujeme proto příkaz **Opatrně Zvedni,** kde Karlovi vysvětlíme jak na to. Výsledný program zapsaný v kopenogramech je v **programu 4.1.** Podívejme se na něj a povězme si, jak jsme jej zapsali.



Program 4.1

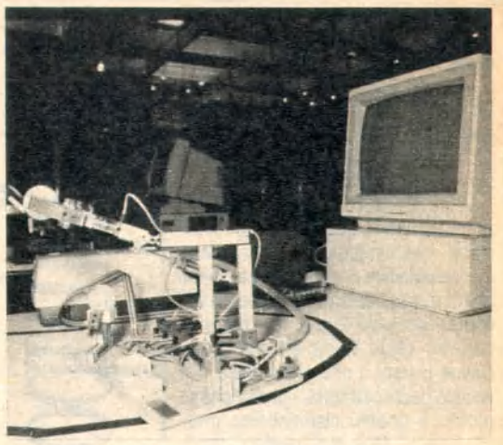
Prvé, co vám jistě padlo do očí, je blok s modrým záhlavím, který jste ještě v našich programech neviděli. Modrá barva je v kopenogramech vyhrazena právě pro rozhodování (mnemotechnická pomůcka: když na křižovatce uvidíte na semaforu svítit modrou, také se zarazíte a začnete přemýšlet). Do záhlaví takovéhoto bloku se pak vepíše podmínka, kterou musí Karel nejprve otestovat a teprve na podkladě výsledku tohoto testu určí, zda část programu zakreslenou uvnitř bloku vykoná či nikoliv.

Co tedy Karel udělá při plnění tohoto programu? Nejprve si přečte podmínku v záhlaví a otestuje ji. V našem případě se nejprve podívá, zda je pod ním značka. Pokud pod Karlem značka je, je podmínka splněna a Karel smí vstoupit dovnitř, do těla podmíněného bloku. Zde jej v našem programu očekává jediný příkaz: **Zvedni.** Karel jej tedy provede a opustí blok a v zápětí i program.

Pokud však podmínka v záhlaví splněna není, Karel do podmíněného bloku při plnění programu vstoupit nesmí. Přeskočí jej tedy a v našem případě to znamená, že neudělá nic.

Co to je, když se řekne Polytechnická pomůcka

1



2

Tato otázka byla v podtextu Výstavy polytechnických pomůcek s mezinárodní účastí, která se konala v PKOJF v Praze ve druhé polovině března. Tuto otázku si nepochybně kladli i pořadatelé, kterých bylo několik, a tak i odpovědí, jež nám nabídli prostřednictvím vystavovatelů, bylo také více. Pokud si návštěvník před vstupem na výstavu představoval pod tímto pojmem například vybavení dílen mladých techniků, pak musel konstatovat, že nemá zdaleka pravdu. Výstava totiž ukázala, že smyslem polytechnické pomůcky je, aby **pomáhala,** nikoli, aby **byla vyráběna.** Jinak řečeno: polytechnická pomůcka je prostředkem, nikoli cílem. Přičemž se od této chvíle nepokládá za podřadné amatérsky vyrábět nejrůznější modely a posléze je využívat jako polytechnickou (učební) pomůcku, jen si musíme uvědomit, že jde o něco jiného.

Polytechnické pomůcky (ve světě) chrlí výrobci ve velkých sériích a ve formě velice jednoduchých stavebnic, z nichž lze sestrojit velmi složité modely téměř všeho, pokud jde o svět techniky pak čehokoli. Slovo model zde není jen synonymem pro hračku, ale platí i jeho původní význam naplněný skutečností, že výsledky na modelu získané lze převést do světa skutečnosti. Předpokladem vzniku takové stavebnice je jednak „vynalezení" co nejjednoduššího a přitom co nejspolehlivějšího spoje dvou stavebních prvků, jednak výroba náročná na přesnost forem a kvalitu materiálu. Tuto cestu zastupovaly na výstavě dvě firmy: Lego a CVK-Fischertechnik. A proč o nich mluvíme na stránce bit/klubu? Protože obě tyto firmy zapojily do hry počítače a patřily k těm málo vystavovatelům, kteří prezentovali počítače jako polytechnické pomůcky a nikoli jako pouhé drahé, byť zajímavé hračky. Obě firmy na výstavě předvedly kromě průřezu svým „hračkářským" programem i výukové systémy, které jsou určeny školám a jiným vzdělávacím institucím a nikoli pro veřejnost. Předpokládají totiž, že budou použity jako praktická ukázka, součást výkladu učitele. Vraťme se však k zapojení počítačů. Technici firmy Lego pro tento účel vytvořili jednoduchý programovací jazyk. Lze jím programovat řízení mechanismů se třemi elektromotory bez předchozích znalostí programování. Děti od 10 let si tak mohou sestavit program, který ovládá např. kosmické vozidlo, jeřáb, montážní linku nebo dopravní signalizaci, pokud místo elektromotorů použijeme světelná návěstidla. Velmi užitečnou aplikaci představuje grafický zapisovač, **obr. 1,** který dokáže zobrazit nejen matematické rovnice, ale i trojrozměrnou skutečnost v axonometrické perspektivě. Speciální interfejs umožňuje připojení k libovolnému počítači.

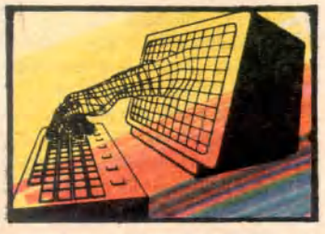
Samozřejmě, že fantazii se meze nejen nekladou, ale její uvolnění je jedním ze smyslů existence takových systémů.

Zatímco Lego pokrývá věkové rozpětí od takřka 1 roku po střední školu a cílem je vzbudit a podchytit zájem dětí zajímavě a snadno dosažitelnými výsledky, má CVK-Fischertechnik celý systém posunutý o 1 stupeň výše. Začíná u dětí předškolního věku a shora není omezen vůbec, protože modelové stavebnice nacházejí uplatnění na vysokých školách i v praxi. Příklad? Model montážní linky ze součástek této stavebnice umožnil odladit program, který pak byl použit pro řízení skutečné linky — v tomto případě u firmy Volkswagen. Podobné modely vlastní různé vysoké školy, které je využívají nejen ke zvýšení názornosti výkladu, ale především k praktickému ověřování technologických projektů, řídicích programů, mechanických vlastností konstrukcí atd. Stavebnicové soupravy jsou vybaveny interfejsem se čtyřmi výkonovými výstupy včetně reverzací, s osmi digitálními a dvěma analogovými vstupy; připojitelný je na výstup Centronix. Na **obrázku 2** je pneumatický robot poháněný malým vysavačem ve funkci kompresoru, který dává přetlak 0,03 MPa. Počítač řídil činnost robotu, která v tomto případě spočívala v přemisťování dvou míčků mezi třemi stanovišti. Poznamenejme zde, že 100 kusů stavebnic různých robotů u této firmy zakoupil pro své počítačové kluby SSM, na jehož stánku byly tyto aplikace vystaveny.

Zmínili jsme se stručně o dvou zahraničních výrobcích, kteří k pojmu polytechnická pomůcka přiřadili obsah, který mu skutečně v současné době náleží. Jejich přínos lze přitom spatřovat ve dvou rovinách: 1) Systémem jednoduché a spolehlivé stavebnice dosahují časových úspor při sestavování modelu, přičemž široká škála prvků umožňuje vysokou variabilnost a značného přiblížení se skutečnosti; 2) Filozofie obou systémů vytváří předpoklady k účinnému překlenutí prostoru od hraní si s počítačem k jeho aktivnímu využití.

(mr)

Foto Mojmír Balling



Počítačová gramatika

8/ Píše Ing. Rudolf Pecinovský, CSc.

Pokuste se nyní sami naprogramovat příkaz **OdstupOdZdi,** při jehož plnění bude Karlovým úkolem zjistit, zda je před ním zeď a v případě, že ano, tak udělat krok zpět. Navrhněte nejprve své vlastní řešení a pak si je zkontrolujte s řešením na obr. 4.2.

Zkusme si nyní další příklad. Naprogramujte příkaz **OpatrnýKrok,** při jehož plnění bude Karlovým úkolem udělat krok, ale pouze v tom případě, že před ním není zeď. Tento příkaz můžeme nadefinovat dvěma způsoby. Buď podle obr. 4.3., tj. tak, jak jsme se doposud naučili, a nebo podle obr. 4.4. V programu na obr. 4.3. jsme museli použít upravené podmínky a ptát se na to, zda před Karlem „není Zeď", v programu podle obr. 4.4. jsme dokonce použili zcela novou konstrukci. Podívejme se nyní na plnění tohoto programu podrobněji.

Karel opět začne tím, že bude testovat podmínku v záhlaví podmíněného bloku. Pokud je podmínka splněna, tj. pokud zjistí, že před ním zeď je, může vstoupit do těla tohoto bloku. Zde však žádný příkaz nenajde, proto jej také ihned opustí a neudělá nic. A to jsme také chtěli.

Pokud však podmínka splněna nebude, tj. pokud před Karlem nebude zeď, vydá se doprava a začne testovat podmínku v druhém záhlaví. Zde najde šipku dolů, což je symbol označující vždy splněnou podmínku — Karel tedy vstoupí do těla této části bloku a zde najde příkaz Krok. Vykoná jej tedy, opustí podmíněný blok a vzápětí ukončí i celý program.

Zopakujeme-li si to ještě jednou, vidíme, že Karel opravdu udělal krok vpřed pouze v případě, že před ním nebyla zeď. V opačném případě neudělal nic.

Obecnější podoba podmíněného bloku, kterou jsme si ukazovali v posledním příkladě, je velice užitečná a mnohé programy by se nám bez ní psaly velice těžko. Představme si například, že Karlovým úkolem bude změnit počet značek na políčku, na němž stojí. Nemůže, bohužel, pouze přidat značku, protože pokud by byl na políčku právě maximální povolený počet značek, způsobilo by přidání další značky chybu. Nemůže ani značku jednoduše zvednout, protože pokud by pod ním žádná značka nebyla, způsobil by pokus o zvednutí neexistující značky rovněž chybu. Nelze použít ani příkazu OpatrněZvedni, který jsme nadefinovali na počátku této kapitoly, protože pokud by pod Karlem nebyla žádná značka, on by neudělal nic a tedy by ani nezměnil počet značek na políčku.

Asi již sami dávno tušíte, že řešení je velice prosté. Pokud pod Karlem bude alespoň jedna značka, tak ji zvedne, a pokud pod ním žádná značka nebude, tak na políčko prostě jednu značku položí. Pokud si na to troufáte, pokuste se naprogramovat řešení sami a svůj program porovnejte s programem na obr. 4.5.

Abyste si však nemysleli, že život je příliš jednoduchý, sdělím vám sladké tajemství, existuje ještě obecnější podoba podmíněného bloku a tedy i podmíněného příkazu (naše bloky jsou vlastně grafickým vyjádřením příkazů v programu). Neexistuje přece žádný pádný důvod pro to, aby podmíněný příkaz měl nejvýše dvě větve. Představte si, že Karlovým úkolem bude zanechat na políčku počet značek, který by jednoznačně určil směr, do nějž je natočen, a udělat dva kroky směrem opačným. Své natočení na východ oznámí prázdným políčkem, natočení na sever jednou, na západ dvěma a na jih třemi značkami. Abyste si ověřili správné pochopení dosavadní látky, pokuste se tuto úlohu vyřešit sami. Kontrolním řešením je pak řešení na obr. 4.6.

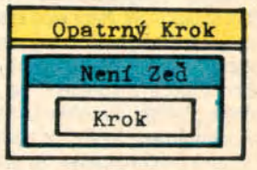
Sami vidíte, že řešení snažící se o maximální efektivitu, nepatří mezi nejprůzračnější. Naproti tomu řešení 4.7. využívající zobecněného podmíněného příkazu je možná trochu upovídanější, ale zároveň je také mnohem přehlednější. Tím se vám vlastně také snažím dát jednu radu:

Nehoňte se ve svých budoucích programech vždy za maximální efektivitou programu. Není zcela vyloučeno, že pokud napíšete program raději přehlednější, ušetříte tím čas nejen sobě, ale i budoucím uživatelům vašich programů.

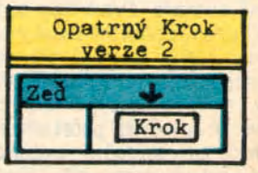
Na závěr kapitoly ještě jeden příklad na procvičení. Představte si, že Karel stojí na kopci, který se svažuje směrem k jihu (kam jinam, že). Naprogramujte příkaz **KrokPoKopci,** v němž budete respektovat tento svah a jím způsobenou obtížnost chůze. Bude-li tedy Karel otočen na sever, tj. do kopce, udělá na tento příkaz pouze jediný krok. Bude-li naopak otočen na jih, tj. z kopce, udělá na tento příkaz kroky tři. No a bude-li otočen na východ a na západ, tj. po vrstevnici, udělá na příkaz KrokPoKopci dva kroky.

Příklad vyřešte jak s ohledem na maximální efektivitu, tak s ohledem na maximální přehlednost výsledného programu. Kontrolní řešení obou programů najdete na obrázcích 4.8. a 4.9.

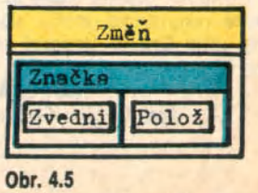
Obr. 4.2



Obr. 4.3



Obr. 4.4



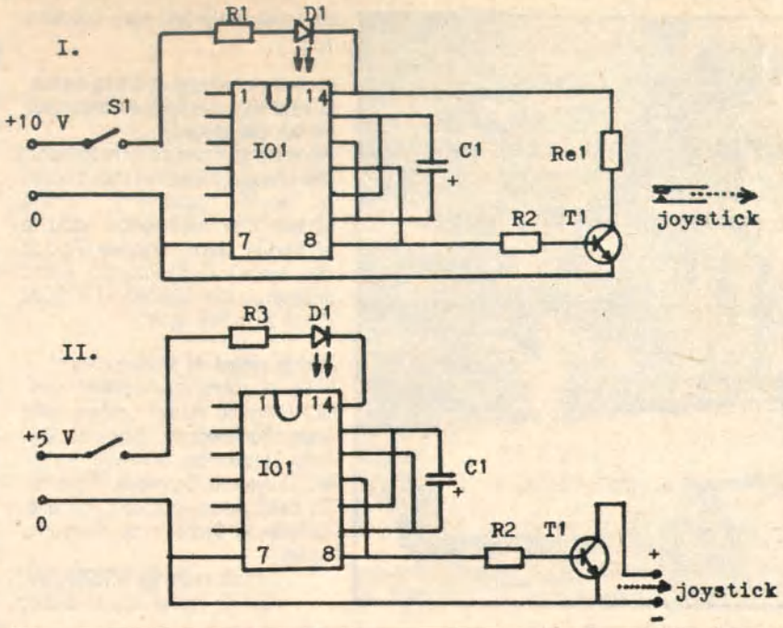
Obr. 4.5

Autofire pre Spektrum

Jedná sa o tzv. AUTOFIRE, teda automatickú streľbu na počítačových hrách. Nápad vznikol z podnetu, že v NSR a Rakúsku predávajú ovládače (joysticky) so zabudovaným AUTOFIRE. Keďže tento ovládač s doplnkom ČSSR nevyrába, ani nedováža prostredníctvom Tuzexu, rozhodol som sa urobiť si takýto doplnok sám a z dostupných prostriedkov. Pri stavbe zariadenia som kládol dôraz na jednoduchosť, účelnosť, minimálnu spotrebu zariadenia a najmä na dostupnosť súčiastok. Kvôli jednoduchosti som pre generátor použil integrovaný obvod MH 7405, teda šesticu invertorov, z ktorého je využitá len polovica. Možno použiť však aj MH 7404, MH 5405, MH 5404. Generátor tvorí okrem IO aj kondenzátor 20 μF a odpor R ≧ 0, tedy vodič (medzi vývodmi 9 a 12 IO). Z toho vyplýva, že frekvencia kmitov sa pohybuje okolo hodnoty 10 Hz. To znamená, že strelec v hre vystrelí 10krát za sekundu. Zvýšiť frekvenciu kmitov možno buď zväčšením kapacity kondenzátora C1, alebo pridaním rezistoru s odporom okolo 10k medzi vývody IO (9, 12). Tranzistor NPN, nízkofrekvenčný, napr. KC 509, ovládaný generátorom spína buď relé (1. varianta), alebo priamo vývody z tlačítka streľby na joysticku (2. varianta). 1. varianta zabezpečuje rýchle kmitanie relátka, ako pri stláčaní tlačítka a tak navodzuje „hráčsku atmosféru". 2. varianta je o ňu ochudobnená, zato je však podstatne lacnejšia. Treba však dbať na polaritu spínaného prúdu v ovládači. Použité súčiastky sú dostupné v každej predajni TESLA Eltos, celé zariadenie v 1. variante vychádza do 100 Kčs, v druhej variante do 50 Kčs.

Tento doplnok som si sám zhotovil na joystick SPECTRA-VIDEO a používam ho na mikropočítači ZX SPECTRUM. AUTOFIRE sa dá pridať vlastne ku každému ovládaču, ktorý má tlačítko na streľbu. Pri troche námahy sa celé zariadenie (bez batérie 4,5 a 9 V) zmestí do krabice joysticku. V tom prípade však pre nedostatok priestoru nemožno použiť plošný spoj.

TOMÁŠ FÜLLÖPP



Rozpis súčiastok:

S1 — spínač
R1 — 220 Ω; TR212
R2 — 2k7; TR212
R3 — 100 Ω; TR212
D1 — LQ 1132; LED
Re1 — relé LUN 5 V
C1 — 20 μF/20 V
T1 — KC 509, KC 508, KC 507
IO1 — MH 7405, MH 7404, MH 5405, MH 5404

INFORMOVAT A BÝT INFORMOVÁN

bylo heslem 5. semináře zájemců o výpočetní techniku ve Svazarmu, který se uskutečnil 18.—20. 3. 1988 v Brně. Celkem 140 účastníků z ČSR, neboť pořadatelem byl ČÚV Svazarmu a jeho brněnské ZO, vyslechlo 28 odborných přednášek a referátů na aktuální témata výpočetní techniky: aplikace systémů CP/M na domácí počítače, sítě počítačů, počítačové jazyky a expertní systémy, efektivnost tvorby programového vybavení, metodika tvorby učebních a demonstračních programů, připojování různých periférií i způsob oceňování tvorby programů dohodou mezi organizacemi. Mezi písemnými materiály bylo oceněno mj. srovnání syntaxe různých BASICů na domácích počítačích rozšířených v ČSSR. Škoda, že z výrobců výpočetní techniky přijela jen Zbrojovka Brno a Didaktik Skalica, aby reprezentovali své výrobky. Podobné semináře pro „značkové" kluby uživatelů osobních počítačů připravují některé aktivní ZO Svazarmu.

Ing. PAVEL HLAVÁČEK

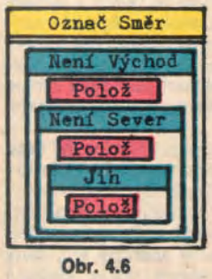
Počítačová gramatika

9/ Píše Ing. Rudolf Pecinovský, CSc.

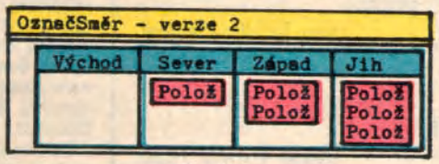
5 Cykly

Mnohokrát jsme již všichni slyšeli, jak jsou počítače neuvěřitelně rychlé. Program, který pro ně pracně připravujeme a kódujeme několik let dokáží vyplnit během několika sekund. S našimi současnými znalostmi bychom nikdy nedokázali připravit pro počítač program, který by jej dokázal zaměstnat ani po několik hodin. V praxi však počítače pracují bez přestávky celé dny a týdny. Avšak ne proto, že by v nich byly tak dlouhé programy, ale proto, že vykonávají některé části programu stále kolem dokola. A o tom, jak to udělat, abychom nemuseli opakující se části programu opisovat, ale abychom uměli počítači vysvětlit, kterou část programu a jak dlouho má opakovat, o tom bude tato kapitola.

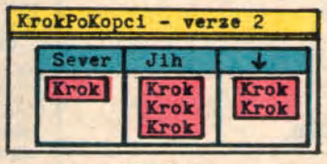
Části programu, které se provádějí opakovaně několikrát za sebou se nazývají **cykly.** U každého cyklu budeme rozlišovat čtyři části **(obr. 5.1):**



Obr. 4.6

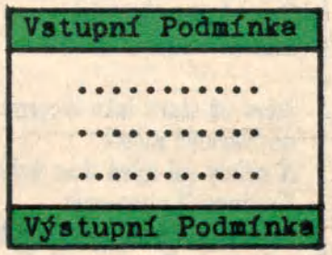


Obr. 4.7



Obr. 4.9

Obr. 4.8



Obr. 5.1

1. Inicializaci — o té prozatím pomlčíme a vrátíme se k ní později.

2. Záhlaví, které obsahuje vstupní podmínku, tj. podmínku, jejíž okamžitá hodnota nám napoví, jestli vůbec smíme do cyklu vstoupit. Je-li splněna, smíme dovnitř. Není-li však splněna, nesmíme tam a musíme pokračovat prvním příkazem za cyklem.

3. Tělo cyklu — což je právě ta část programu, která se má opakovaně provádět.

4. Zápatí, které obsahuje výstupní podmínku, tj. podmínku, jejíž okamžitá hodnota nám poví, jestli již smíme cyklus opustit. Je-li splněna, smíme ven a pokračujeme prvním příkazem za cyklem. Není-li však splněna, musíme se vrátit a provádět celý cyklus znovu. Podtrhuji celý cyklus, tj. včetně testování vstupní podmínky v záhlaví.

S volbou pokračování v závislosti na hodnotě nějaké podmínky jsme se tedy setkali ve třech souvislostech: v záhlaví podmíněného příkazu a v záhlaví a zápatí cyklu. Možná, že jste si všimli, že vyhodnocení směru dalšího postupu je v podstatě jednotné. Pro jistotu však připomenu, že:

při volbě směru dalšího postupu na základě vyhodnocení nějaké podmínky vždy platí:

SPLNĚNO = DOLM
NESPLNĚNO = Druhým v úvahu přicházejícím směrem.

V některých cyklech nemusí být vstupní nebo výstupní podmínka anebo ani jedna z nich podmínkami v pravém slova smyslu. Mohou to být logické konstanty s hodnotou ANO nebo NE stejně, jako jí byla šipka dolů, s níž jsme se seznámili v podmíněném příkazu. O takovýchto konstantních podmínkách budeme říkat, že jsou degenerované. Jednotlivé druhy cyklů potom rozlišujeme podle toho, které z podmínek jsou plnohodnotné a které degenerované.

5.1 Nekonečný cyklus

Ukažme si vše na příkladech. Pro začátek si povíme o nejjednodušší variantě cyklu s oběma podmínkami degenerovanými. Takovýto cyklus mívá vstupní podmínku vždy pravdivou a výstupní podmínku naopak vždy nepravdivou. Vstupní podmínka nás tedy vždy pustí dovnitř a výstupní podmínka nás naopak nepustí nikdy ven. Takovýto druh cyklu se nazývá nekonečný cyklus. Nekonečný proto, že jej nikdy nemůžeme opustit, a že proto tělo cyklu provádíme donekonečna kolem dokola.

Nemyslete si, že nekonečný cyklus je nějaká abstrakce. Naopak. Je často používaný. Prakticky v každém počítači běží nějaký nekonečný cyklus (někdy dokonce hned několik). Zapnete-li počítač, proběhne v něm nejprve inicializační proces, načež od vás počítač očekává příkaz, vyplní jej a očekává další příkaz, který opět vyplní, a tak to jde stále dokola, dokud jej nevypnete anebo (což je u tuzemských počítačů bohužel až příliš často) dokud se neporouchá.

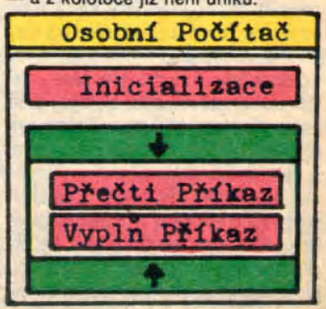
Kdybychom chtěli tento cyklus znázornit v kopenogramu: obdrželi bychom program **Osobní Počítač — obrázek 5.2.** Podívejme se nyní na něj a vysvětleme si jej.

První, čeho jste si asi již všimli na obrázku 5.1 je, že záhlaví a zápatí cyklu vybarvujeme zeleně. Tím je odlišíme od podmíněných bloků i prostých příkazů.

V záhlaví nalézáme šipku dolů. Jak si jistě pamatujete, používali jsme ji již v kapitole o podmíněných příkazech a symbolizuje nám vždy splněnou podmínku.

Jelikož je tedy vstupní podmínka splněna, smíme vstoupit do těla cyklu. Zde nás očekávají dva příkazy: načtení příkazu operátora a jeho vyplnění.

Po provedení těla cyklu nás otevřenou náručí očekává zápatí s výstupní podmínkou. Zde je symbol, s nímž jsme se dosud nesetkali — šipka vzhůru. Jak si asi domyslíte, na rozdíl od šipky dolů nám bude šipka vzhůru symbolizovat podmínku vždy nepravdivou. Jelikož tedy výstupní podmínka splněna není, nemůžeme cyklus opustit a musíme se vrátit znovu na začátek — tj. na testování podmínky v záhlaví. Ta je samozřejmě vždy splněna a pustí nás tedy dovnitř — a z kolotoče již není úniku.



Obr. 5.2

Počítač áno a čo ďalej? II

Pozrime sa na počítače. Aké sú oni dnes, aké služby poskytujú a ako sme s nimi spokojní.

Tak, ako rastie počet druhov počítačov, ako rastie ich množstvo, úmerne k tomu rastie počet ľudí, ktorí majú možnosť sa s nimi stretnúť a urobiť si vlastnú predstavu na ne i na to, na čo by počítač potrebovali.

Počítač, môj fenomén! Si stále rýchlejší, rastie počet informácií, ktoré urobíš za sekundu, rastie tvoja operačná i vonkajšia pamäť, rastie tvoja programová podpora. Si stále spoľahlivejší, si stále menší. Čoraz dokonalejšie vieš so mnou komunikovať, prestávaš byť závislý na klimatizácii, nepotrebuješ mať pri sebe stále technika, aby ťa udržiaval v chode, si jednoducho čím ďalej tým naj... naj... naj...

Napriek tomu, že to, čo som povedal, je pravda, vráťme sa pekne na zem. Pozrime sa okolo seba a vidíme že...

Výpis zo sporožírového účtu pekne úhľadne vytlačí počítač. Pozrieme do neho a ajhľa, multiservis nezaplatený, zrážka z platu nezaevidovaná. Ideme do banky. „Čo sa rozčuľujete, došlo k chybe v počítači, my o nej vieme, zrážku vám na budúci mesiac započítame. Aj keď máte trvalý príkaz na platbu multiservisu, vyplňte si jednorazový príkaz, veď viete ako je to s počítačmi!"

Prišiel vám zlý telefónny účet? Samozrejme je to vaše subjektívne hľadisko. „Nepochopili ste zrejme dobu, veď ho vystavil počítač!"

Idete si zvýšiť zálohu na odber elektrickej energie. Prídete v úradných hodinách, máte príslušné doklady, vyplníte tlačivo a slušne vám oznámia, že o pol roka budete mať zálohu v rámci združeného inkasa zvýšenú. Prečo až o pol roka, čudujete sa. „Veď viete, ide to cez počítač," zaznie spiklenecká odpoveď.

Je na vine počítač? Iste, pokiaľ je na vine vysávač, že slabo ťahá, lebo ste mu zabudli vyprázdniť vrecko.

Príklady, ktoré som vybral, nie sú celkom výstižné, ale skúsme hľadať príčinu neuspokojivého využívania počítačov. V prvom rade sa natíska otázka. A možno máme zlé počítače. Iste, máme rozličné počítače. Sú rôzne výkonné, rôzne spoľahlivé z technického pohľadu a máme ich zatiaľ málo. Žiaľ, prakticky ešte menej, pretože miesto jedného spoľahlivého musíme mať povedzme dva menej spoľahlivé, aby sme udržali informačný alebo riadici tok informácií pri živote. No nech je situácia akokoľvek zložitá, naše počítače, či lepšie povedané tie ktoré máme, sú schopné poslúžiť nám rovnako kvalitne s rovnakou presnosťou, ako iné vo svete. Iste, prácnosť je vyššia, možnosť porúch je vyššia, ale v konečnom dôsledku ich technická nespoľahlivosť nie je na príčine zlého nasadenia, nespokojnosti s nimi. Do určitej miery má však aj tento fakt vplyv na to, že sa podstata zahmlieva.

Dohodli sme sa, že počítače máme také ako inde, aj keď s pripomienkami. Pátrajme ďalej. Čo vlastne núti počítač robiť to, čo robí? Na súhrn pokynov, inštrukcií, ktoré má počítač vykonať, sa zaužíval pojem program.

Máme teda zlé programy? Ako vlastne vznikajú, nie je tam chyba?

Iste, programy môžu byť rýchlejšie i pomalšie, pri tom istom algoritme možno do nich vniesť aj zbytočnú činnosť počítača. Ale nie je pravda, že by nám nerobili to, čo chceme, to teda nie. Programátor dostane podklady o tom, čo má byť cieľom programu, ako má ten cieľ dosiahnuť a na základe akých vstupov. Jeho správnosť zadávajúci skontroluje a potvrdí. Našli sme ďalšieho v reťazi, čiže toho, ktorý hovorí, čo má vlastne program robiť a ako.

Tu sme sa dostali tam, kde sme chceli byť. Už sme tak ďaleko od počítača, že je zrejmé, že hľadáme príčinu tam, kde je. Ten, kto rozhoduje o tom, ako vlastne počítač plní naše požiadavky, je človek. Nepátrajme ďalej po vinných, alebo nevinných. Pozrime sa, ako rôzny pohľad človeka na využívanie počítačov, na ich funkčnosť, môže priviesť k rôznym výsledkom.

Zoberme si za základ moment, kedy človek siahne po počítači napríklad v oblasti riadenia a ako jeho postoj ovplyvní následný prínos počítača v živote — praxi.

Mám obrovský riadiaci aparát, množstvo výkazov, prepočtov, v ktorých sú často chyby. Všetko mi chodí neskoro, musím ich však dostávať, musím ich odovzdávať. Rozhodujem, ale prakticky len intuitívne. Takto to ďalej nemôže ísť, veď máme počítače. Vybavím si počítač, ľudí, ktorí sú na to viac alebo menej povolaní a teraz poďme na to. Urobte niečo, aby mi počítač pomohol.

Prvý pohľad.

Tu máte počítač, vidíte aká je situácia, dajte mi to na počítač. Urobí sa analýza, navrhnú sa programy a celý chaos sa dostal do počítača. Čo som získal? V prvom rade potrebujem ďalších ľudí k počítaču, lebo nakŕmiť ho všetkými potrebnými údajmi je sizyfovská práca. Dostávam obrovské množstvo rôznych zostáv, v ktorých sa topím a zväčša tiež neskoro. Problémy mi neubudli, iba rastú, pôvodné ťažkosti sa prehlbujú, nastáva rozčarovanie z počítača. Prijímam ďalších ľudí, žiadam zmeny, vylepšenie, alebo sa vzdávam, odchádzam od počítača, zavrhujem ho.

Druhý pohľad.

Tu máte počítač, rozoberme si situáciu. Zistite čo ju zapríčiňuje, navrhnite, ako ju najlepšie zvládnuť. Určite úlohu, ktorá pripadne počítaču. Urobí sa analýza, určí sa nevyhnutný okruh údajov, minimalizuje sa okruh výstupných údajov, zabezpečí sa ich dostupnosť vždy vtedy, keď ich potrebujeme. A ajhľa, je ten počítač na niečo, i keď to nie je zatiaľ žiadna sláva. Každopádne sa mi uľavilo, môžem riešiť ďalšie úlohy. Samozrejme trvám na tom, aby počítač bol mojím pomocníkom a on sa ním čím ďalej tým viac aj stáva.

Ponúkol som dva extrémy. Prvý sa snaží preniesť súčasný chaos na vyriešenie počítača. To ale počítač nie je schopný. Druhý odhaľuje príčinu chaosu a snaží sa ju odstrániť a pritom určuje miesto počítača v systéme riadenia tak, aby bol schopný pomáhať pri riadení. Možno mi dáte za pravdu, že iba druhý spôsob vedie k cieľu — že nasadenie počítačov bude na požadovanej úrovni. Táto myšlienka, napriek tomu, že nie je nijako nová, si cestu životom razí veľmi pomaly. V súčasnosti, vzhľadom na prestavbu celého hospodárskeho mechanizmu získava na aktuálnosti. Rozhodovanie, akou cestou sa vybrať alebo pokračovať, je len v rukách ľudí a nikdy nie počítačov.

Ing. VLADIMÍR LABÁTH

Počítačová gramatika

10/ Píše Ing. Rudolf Pecinovský, CSc.

5.2. Cyklus se vstupní podmínkou

Podívejme se nyní na rafinovanější podoby cyklu. Prvním z nich bude cyklus se vstupní podmínkou plnohodnotnou a výstupní podmínkou degenerovanou (vždy nepravdivou) nazývaný zkráceně cyklus se vstupní podmínkou.

Naprogramujme příkaz KeZdi, na nějž Karel dojde ve směru, do nějž je natočen, až k nejbližší zdi. Pokusili-li bychom se vyjádřit příkaz slovy, asi bychom Karlovi řekli: „Jdi až ke zdi!" Tomu by však Karel asi vůbec nerozuměl. Vyjádřili bychom se tedy přesněji: „Dělej kroky dokud se před tebou neobjeví zeď!" Je nám však jasné, že Karel se musí vždy nejprve podívat, jestli před ním není zeď, pokud ne, tak udělat krok, znovu se podívat, a tak dále, dokud se před ním zeď neobjeví. Pak může všeho nechat, protože je s prací hotov. V kopenogramech bychom řešení zakreslili podle **obr. 5.2.1.**

Všimněte si jedné význačné vlastnosti cyklu se vstupní podmínkou:

je-li vstupní podmínka hned na počátku nepravdivá, neprovede se tělo cyklu ani jednou!

Pokuste se nyní sami naprogramovat příkaz Vyber, na nějž má Karel pod sebou vybrat všechny značky a příkaz NaSever, na nějž se má Karel otočit na sever. Vzorovými řešeními jsou programy **na obr. 5.2.2 a 5.2.3.**

5.3. Cyklus s výstupní podmínkou

Dalším druhem cyklu je cyklus s degenerovanou (vždy pravdivou) vstupní podmínkou a plnohodnotnou podmínkou výstupní, nazývaný zkráceně cyklus s výstupní podmínkou.

Ukažme si jeho činnost opět na příkladu. Představte si, že Karel stojí kdesi na dvorku a může pod ním být i značka. Jeho úkolem je dojít na další značku aniž by změnil počet značek na pozici, na níž právě stojí. Předpokládáme, že mezi Karlem a zdí opravdu nějaká značka je. Pokud bychom pro tuto úlohu chtěli využít cyklu se vstupní podmínkou, museli bychom navrhnout řešení podle **obrázku 5.3.1.** Daleko elegantnější je však řešení prostřednictvím cyklu s výstupní podmínkou tak, jak je předvedeno na **obrázku 5.3.2.**

Z uvedeného příkladu jste jistě postřehli důležitou vlastnost cyklu s výstupní podmínkou, a to, že

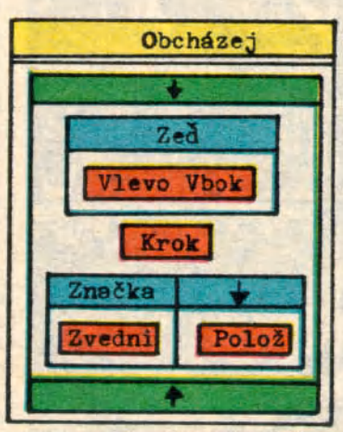
Tělo cyklu s výstupní podmínkou se provede vždy alespoň jednou!

Na závěr podkapitoly opět příklad k procvičení. A protože jste již trochu pokročilí, bude to příklad trochu těžší. Karel stojí kdesi na dvorku a není pod ním žádná značka. Na některém políčku před ním značka je a Karlovým úkolem je poponést tuto značku o políčko blíže ke své výchozí pozici a vrátit se do výchozí pozice. Pokud se značka nachází hned na sousedním políčku, Karel ji pouze zvedne a vrátí do výchozí pozice. Z cvičných důvodů nepoužívejte proceduru NaDalšíZnačku. Vzorové řešení je **na obr. 5.3.3.**

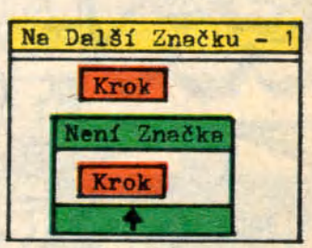
Obr. 5.2.1

Obr. 5.2.2

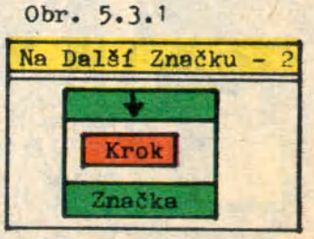
Obr. 5.2.3



Obr. 5.3 Na vyhnutí se obvodové zdi jeden Vlevo Vbok stačí



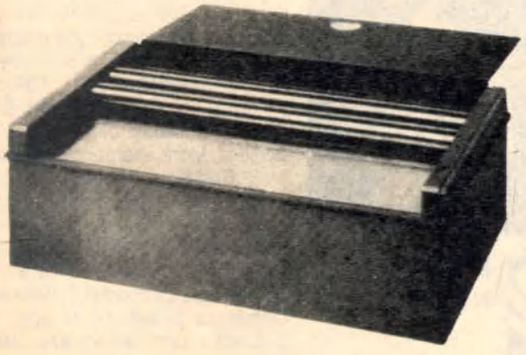
Obr. 5.3.1



Obr. 5.3.2

Tuzemské tiskárny pro osobní počítače

Většina majitelů počítače si jej koupila proto, aby mohla hrát hry. Většinou tedy šlo o nákup drahé hračky pro celou rodinu. Někteří se časem nasytí her a začnou se zajímat o to, co lze s počítačem dělat dál. Významnou roli v tomto myšlenkovém posunu hrají svazarmovské kluby, kam se nový majitel záhy dostaví, aby získal programy pro svůj počítač. Zde zjistí, že mnozí počítačoví fandové nejen že hry vůbec nehrají, ale navíc spolu mluví jazykem, z něhož laik není chytrý. Pochytí, že existují také uživatelské programy, které naplňují původní smysl existence počítače, ale že jejich rozšíření brání nedostatek periferií zvaných tiskárny, které umožňují trvale zachytit výsledky práce počítače. Nedávná pražská výstava polytechnických pomůcek (viz též SP č. 8/88 str. 24) nabídla možnost podívat se, jaká je současná situace a co nás v tomto oboru čeká v budoucnosti. Vám, kteří jste se nemohli výstavy zúčastnit přinášíme nyní stručnou zprávu z této oblasti.



BT 100

Nedostatek tiskáren řešili majitelé počítačů doposud soukromým dovozem (tuto možnost nemá každý), nákupem tiskáren dovezených státním obchodem (nedávno např. plotter Sharp), úpravami vyřazených dálnopisných tiskáren a konečně i vlastní výrobou tiskárny. Žádný ze zmíněných způsobů nedostatek tiskáren nevyřešil, kromě způsobu posledního, který k řešení, paradoxně, alespoň významně přispěl. Amatérské tiskárny se objevily na výstavách, kde si jich všimli potencionální profesionální výrobci. V případě tiskárny BT 100 to byli pracovníci Tesly Přelouč. Nápad upravit pro sériovou výrobu sice amatérský, ale hlavně již vyvinutý výrobek je cenný zvláště proto, že se přece jenom krátí doba uvedení výrobku na trh. Tiskárna se bude prodávat jednak samostatně, jednak v sestavě se stolním magnetofonem (datarecorderem) SP 100. Původně ohlášený termín prodeje na konec dubna byl posunut na červenec.

Technické údaje tiskárny BT 100:

Dvoumotorový pohonný systém	
Rychlost tisku	150 bodů/sec. (4 znaky ASCII)
Jednobodový tisk na papír formátu A4 podložený úhlovým papírem.	
Softwareové ovládání z počítače v úrovni TTL	
Komunikační rozhraní	4 bity vstupní, 4 bity výstupní
Napájení	24 V=, odběr špičkově 3A
Rozměry	240 × 150 × 75 mm
Hmotnost	asi 1,5 kg
Předpokládaná cena	MC 1600 Kčs bez datarecorderu
Výrobce	koncernový podnik Tesla Přelouč

MERKUR ALFI

Pokud se vám po přečtení názvu vybavila dětská stavebnice Merkur, pak jste uhádli. Výrobce Kovopodnik PMH Broumov zvolil stejnou filozofii jako firmy Lego a Fishertechnik a využil zavedených prvků, které doplnil několika upravenými kusy, ke kompletaci stavebnice souřadnicového zapisovače. Použité krokové motorky jsou ve všech fázích řízeny mikropočítačem, což umožnilo vyhnout se použití logických obvodů. Z toho vyplývá jednoduchost desky elektroniky a napájecího zdroje. Technickou úroveň dokumentuje fakt, že zařízení dokáže kreslit alfanumerické znaky od minimální výšky písma 1,2 mm (!). Dané programové zařízení (upravené zatím pro mikropočítač ZX Spektrum) umožňuje využívat toto zařízení jako jednoduchou tiskárnu (printer), dále jako klasický souřadnicový zapisovač (plotter), samozřejmostí je možnost vytváření grafických kopií obrazovky. Po úpravě zapisovací hlavy může zařízení (díky použití krokových motorů) sloužit i pro digitalizaci obrazových předloh (jako scanner), což znamená obrácený postup, tj. převedení kresby na číselné hodnoty a jejich uložení do paměti počítače. Sluší se dodat, že stavebnice byla vyvinuta ve spolupráci s ODPM v Prešově, konkrétně s ing. Vladimírem Dovalem. Dotazem u výrobce jsme zjistili, že do konce roku by měla být vyrobena ověřovací série. Další osud výroby zapisovače a jeho distribuce je předmětem intenzívního jednání.

Technické údaje zapisovače MERKUR ALFI:

Pohon	dva krokové motory
Rychlost v režimu printer	cca 3 znaky/sec.
Rychlost posuvu v ose x i y	cca 50 mm/sec.
Jednobodový tisk pomocí vloženého písátka (fix apod.)	
Minimální velikost kroku	cca 0,15 mm
Formát papíru	A4
Softwareové ovládání z počítače v úrovni TTL	
Komunikační rozhraní	9 vstupů
Napájení	8 V=+/—10 %, odběr 1,1 A
Rozměry	355 × 100 × 75 mm
Hmotnost	1,2 kg
Předpokládaná cena	cca 1000 Kčs
Výrobce	Kovopodnik PMH Broumov provozovna 02 Police nad Metují

GAMA 01

Na závěr se zmiňme ještě o dvoujehličkové tiskárně, která již slouží řadě majitelů. Autorem prototypu, který užívali členové kroužku při DPM v Berouně, je ing. Birka, s nímž výrobce Gama Milevsko i nadále spolupracuje. Tiskárnu již vlastní řada zájemců, kteří si ji objednali po jejím uvedení na výstavě Zenit před dvěma roky. V době pražské výstavy byla výroba přerušena. Ne, nelekejte se, pouze přerušena, nikoli zastavena. Po nabytí prvních zkušeností s výrobou tiskárny byly jednak zaváděny technické změny, jednak probíhalo předzásobení materiálem pro sériovou výrobu. Do konce roku by mělo být uspokojeno dalších asi tisíc žadatelů z celkového počtu několika tisíc objednávek registrovaných výrobcem. O způsobu distribuce prostřednictvím obchodní sítě se dosud jedná.

Technické údaje tiskárny GAMA 01:

Rychlost tisku	25 znaků/sec.
Velikost písma	2,5 mm při rastru 5×7
Počet jehliček	2, jedna tiskne zleva do středu, druhá od středu doprava
Formát papíru	A4, jedn. listy (max. 4 kopie) leporelo do A4, dálnopisná role
Napájení	220 V; —10, +15 %; 50 Hz
Rozměry	415 × 150 × 90 mm
Hmotnost	5,5 kg
Cena	VC 3680 Kčs, MC 4990 Kčs
Výrobce	Gama Milevsko

(bm)
Foto Mojmír Balling a archiv

Počítačová gramatika

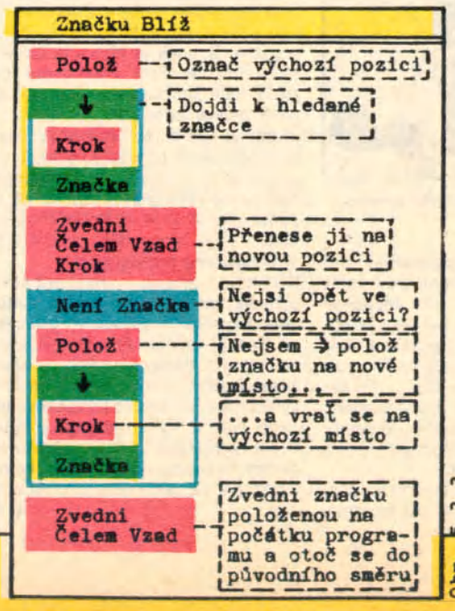
11/Píše ing. Rudolf Pecinovský, CSc.

5.4. Cyklus s oběma podmínkami

Přejděme nyní k nejrafinovanější variantě cyklů, kterou jsou cykly s oběma plnohodnotnými podmínkami. Tato varianta je natolik rafinovaná, že v řadě programovacích jazyků ani nejde naprogramovat, a proto se o ní zmíním opravdu jen stručně.

Navrhněme rychlejší variantu procedury NaDalšíZnačku, v níž budeme vyšetřovat i případ, že mezi Karlem a zdí vlastně vůbec žádná značka být nemusí. V tom případě bychom museli před každým krokem testovat, jestli před Karlem není náhodou zeď a pokud ano, tak cyklus ukončit. Jedná se tedy vlastně o sloučení programů KeZdi a NaDalšíZnačku. Výsledným programem pak bude program na **obr. 5.4.1.**

K úlohám, vedoucím na takovýto druh cyklů se ještě vrátím až budeme trochu chytřejší. Povíme si pak, jak je možno takový cyklus převést na cyklus s výstupní podmínkou.



Obr. 5.3.3

Obr. 5.3.2

Obr. 5.4.1

Sloupkový diagram

Program, jehož výpis uvádíme umožňuje graficky znázornit sled činností, kde na ose X může být až 12 sloupců (činností) a na ose Y jejich hodnoty do výše 10 000. Program je určen pro počítače COMMODORE C116, C16 a PLUS 4.

V řádcích 120 až 140 je omezení maximální výšky sloupců. Rozpočítání šířky sloupců podle jejich počtu provádějí řádky 150 až 170. V řádku 160 je použit příkaz SCALE v modu 1. Tento mod umožňuje přijímat do souřadnic X a Y hodnoty 0 až 1023. Jemnější dělení souřadnic neodpovídá vyššímu fyzikálnímu rozlišení na obrazovce, ale interně jsou hodnoty přepočteny na reálné souřadnice. Zde je tohoto modu s úspěchem využito k automatickému rozdělení šíře sloupců podle počtu zadaných dat. Řádky 180 až 210 kreslí vodorovné linky grafu. Jejich popis, zjednodušeně jen polovinou a maximem, provedou řádky 220 až 270. Vlastní kreslení sloupců odpovídajících výši zadaných dat provádějí řádky 280 až 350. Hodnoty, které chceme zobrazit, vložíme na řádky 380 a 390 příkazem DATA. Stiskem kterékoliv klávesy lze program ukončit a přejít do BASICu. **(MB)**

```
100 REM * SLOUPKOVY DIAGRAM *
110 TRAP160:GRAPHIC 1,1
120 REM * OMEZENI MAXIMA *
130 READ D:IF D>M THEN M=D
140 N=N+1:GOTO130
150 REM * ROZPOCITANI SOURADNIC *
160 SCALE1:S=INT(.5+LOG(M)/LOG(10))
170 Y0=10^(3-S):X0=1024/(N+3):E=S
180 REM * KRESLENI LINEK *
190 FOR I=0 TO 10
200 DRAW1,0,I*100 TO 1023,I*100
210 NEXT
220 REM * POPIS LINEK *
230 P=INT(10^E)
240 P$=MID$(STR$(P),2,LEN(STR$(P)))
250 CHAR1,0,0,P$:P=P/2
260 P$=MID$(STR$(P),2,LEN(STR$(P)))
270 CHAR1,0,12,P$
280 REM * KRESLENI SLOUPCU *
290 RESTORE:FOR I=1 TO N
300 READ D:X=I*X0
310 Y=1000-(D*Y0):IF Y<0 THEN Y=10
320 BOX1,X+X0,Y,X+X0*1.75,1000,,1
330 B=(X+X0)*320/1024:B=INT(B/8+.5)
340 P$=MID$(STR$(I),2,LEN(STR$(I)))
350 CHAR1,B,24,P$:NEXT
360 GETKEYK$:GRAPHIC0,1
380 DATA 3300,9512,7000,8000,6543,7531
390 DATA 8521,1523,3214,4569,7894,6542
```

Oba výpisy programů jsou vytisknuty na tiskárně, která znak dolar píše '$' a znak umocnění (šipka nahoru) píše '^'

Hlasitá klávesnice počítačů COMMODORE

```
10 REM"PIPNUTI KLAVESY"
20 FOR I=0 TO 46
30 READ A$:A=DEC(A$):POKE 1630+I,A
40 B=B+A:NEXT
50 IF B=5208 THEN SYS1630
60 ELSE PRINT"CHYBA V DATECH!"
70 END
80 DATA 78,A9,7F,8D,10,FF,A9,B0,8D,0F
90 DATA FF,A2,75,A0,06,8E,14,03,8C,15
100 DATA 03,58,60,A5,C6,C9,40,F0,08,A9
110 DATA 22,8D,11,FF,4C,0E,CE,A9,00
120 DATA 8D,11,FF,4C,0E,CE,00,00
```

U nejmenšího typu COMMODORE C116 je tak zvaná „gumová" klávesnice, u které není jednoznačný pocit stisku klávesy. Popsaný program zajistí pípnutí při každém stisku kterékoliv klávesy, kromě klávesy CONTROL, COMMODORE a SHIFT. Je použitelný pro počítače C116, C16 a PLUS 4.

Program je monitorový (strojový), napsaný a spouštěný v BASICu. Cyklus FOR...NEXT čte DATA a ukládá je od adresy 1630. Řádky 50 a 60 provedou kontrolu správného načtení dat kontrolním součtem. Program spustíme příkazem RUN a po příkazu NEW můžeme načíst další program, který chceme editovat a klávesy se nám stále ozývají, až po vypnutí počítače. Po případném použití tlačítka RESET obnovíme pípání příkazem SYS1630. **(MB)**

Listárna

V čísle 2 Světa práce jsme vás na tomto místě vyzvali ke spolupráci. Tato výzva stále platí. Očekáváme, že nám napíšete, jak vám tato rubrika slouží, co byste si zde rádi přečetli, popř. pošlete i svůj příspěvek. (Příspěvky, které otiskneme, také honorujeme.) Takto jsme již otiskli stavební návod na stavbu stykového obvodu (interfejs) pro kopírování programů v č. 6, dále stykového obvodu k magnetofonu od ing. Miroslava Boháče v č. 7, Autofire pre Spektrum od Tomáše Füllöpa a zprávu o semináři od ing. Pavla Hlaváčka v č. 9. Všem těmto autorům děkujeme a doufáme, že budou spolupracovat i v budoucnu. Připomeňme si ještě, že příspěvky k otištění je nutné psát na stroji, abychom co možná vyloučili chyby při jejich přepisu z rukopisu, obrázky i jejich popis musí být čitelné. Upozorňujeme také, že za správnost a úplnost údajů ručí autor.

Píší nám však i čtenáři, kteří žádají o radu. A tak jsme se v posledním období zajímali o to, co mají dělat majitelé počítačů z míst, kde nejsou značkové kluby. Potěšitelnou odpověď na tuto otázku nám dal ing. Bukovský ze 602. ZO Svazarmu, která sdružuje největší počet značkových klubů. Jejími členy byli dříve i mimopražští majitelé počítačů, dokud počet všech členů byl zvládnutelný. Nyní existuje řada klubů v krajích a okresech. Napíšete-li si na adresu **602. ZO Svazarmu, Zikmunda Wintra 8, 160 00 Praha 6,** dostanete odpověď s adresou vám nejbližšího značkového klubu. Možná ale, že byste získání takové informace mohli urychlit dotazem na okresním nebo krajském výboru Svazarmu a teprve po případném neúspěchu se obrátit na uvedenou adresu.

Na zájmové kroužky radioamatérů a elektroniků Svazarmu nebo při ZK ROH musíme odkázat i ty čtenáře, kteří nás žádali o podrobnosti ke stavbě návodů uveřejněných na této stránce. Jejich uveřejněním jsme chtěli pomoci při rozšiřování technického vybavení. Nelze však, vzhledem k rozsahu strany, poskytovat elementární informace. Nezbývá, než předpokládat určitou odbornou zdatnost našich čtenářů, nebo alespoň jejich schopnost navázat kontakty s příslušnými odborníky.

Tak tedy nezapomeňte:

BIT/KLUB JE TU PRO VÁS

Redakce

Počítačová gramatika

12/ Píše Ing. Rudolf Pecinovský, CSc.

NOVĚ DEFINOVANÉ PODMÍNKY

Kdykoliv jsme se doposud rozhodli Karla něco naučit, nadefinovali jsme nový příkaz. Programátoři by řekli, že nadefinovali novou proceduru.

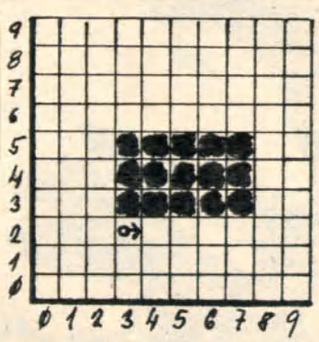
Jak však již víme, Karel umí od narození nejen plnit čtyři primitivní příkazy **(Krok, VlevoVbok, Polož, Zvedni),** ale umí zároveň testovat i šest až sedm primitivních podmínek **(Zeď, Značka, Sever, Východ, Západ, Jih** a někdy i **Stisk auto),** nebo, řečeno „programátorsky", logických funkcí, tj. funkcí, které nám vracejí logickou hodnottu ANO nebo NE, PLATÍ či NEPLATÍ apod.

Jistě si dovedete představit, jak špatně by se nám programovalo, pokud bychom museli vystačit pouze s primitivními příkazy a nemohli si dodefinovat příkazy (procedury) vlastní. Skoro stejně nepříjemná je i nemožnost nadefinovat si vlastní podmínky — logické funkce.

Než však přejdeme k vlastnímu výkladu definice logických funkcí, musíme si nejprve objasnit jeden termín z předminulého odstavce. Co to znamená, že funkce vrací hodnotu? Podívejte se ještě jednou na naše příklady z minulé a předminulé kapitoly. Vždy, když jsme potřebovali vědět, zda je nějaká podmínka splněna (např. zda je Karel otočen na sever), zavolali jsme patřičnou funkci a nechali jsme si to zjistit. Dotyčná funkce zjistila pravý stav věci a **vrátila hodnotu,** kterou jsme pak otestovali. Říkali jsme sice, že testujeme podmínku, ale ve skutečnosti jsme **testovali hodnotu, kterou nám vyvolaná funkce vrátila.** Další chod programu pak závisel na tom, zda nám vrátila hodnotu **ANO** a nebo hodnotu **NE.**

Ukažme si nyní na příkladu, jak bychom mohli takovou vlastní logickou funkci využít v programu. Představte si, že kdesi uprostřed Karlova dvorku stojí zděná garáž s obdélníkovým půdorysem (obr. 6) a Karel má za úkol tuto garáž neustále obcházet kolem dokola. Pokud předpokládáme, že Karel již stojí připraven u stěny garáže pak bychom obcházení proti směru hodinových ručiček nadefinovali např. podle obrázku 6.1.

V záhlaví cyklu se vstupní podmínkou máme podmínku (funkci), kterou Karel doposud nezná a kterou budeme muset teprve nadefinovat. Ukážu vám dvě možnosti nadefinování logické funkce **VlevoZeď.** Obrázek 6.2 ukazuje variantu, která je sice na první pohled složitější, ale pro většinu z vás bude pochopitelnější.



Obr. 6

Obr. 6.1

Obr. 6.2

Obr. 6.3

USERPORT displej

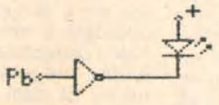
```
10 FOR A=49161 TO 49152+167:READ B
20 S=S+B:POKE A,B:NEXT
30 PRINT"□":SYS 49309
60 DATA 169,48,157,30,4,169,0,157,30,216,173,1
80 DATA 221,234,96,169,192,141,1,192,173,1,221,162
90 DATA 0,160,46,140,0,192,41,128,208,98,32,9
140 DATA 192,232,160,59,140,0,192,41,64,208,85,32
150 DATA 9,192,232,160,72,140,0,192,41,32,208,72
160 DATA 32,9,192,232,160,85,140,0,192,41,16,208
170 DATA 59,32,9,192,232,160,98,140,0,192,41,8
180 DATA208,46,32,9,192,232,160,111,140,0,192,41
190 DATA 4,208,33,32,9,192,232,160,124,140,0,192
200 DATA 41,2,208,20,32,9,192,232,160,137,140,0
210 DATA 192,41,1,208,7,32,9,192
220 DATA 76,49,234,234,169,49,157,30,4,169,0,157,30,216,173,1
230 DATA 221,108,0,192,169,24,141,20,3,169,192,141
240 DATA 21,3,96
260 REM *** USERPORT DISPLEJ COMMODORE C 64  ***
270 REM POKE 56579,255 = VYPNUTO

READY.
```

Při používání hardware a jeho zkoušení je dobré používat USERPORT displej. Co to je? USERPORT displej může být buď hardwarový nebo softwarový a slouží k zobrazení stavu na datové sběrnici počítače. Hlavní využití je při odlaďování programů pro řízení hardware, a to bez jeho použití, a nebo při hledání chyb, pro rozlišení zda se chyba nachází v programu, nebo v připojeném zařízení.

Před zapnutím počítače zasuneme modul do USERPORTu, nebo naloudujeme do počítače před vlastním zkoušeným programem program USERPORT DISPLEJ. Potom se při zkoušení programu zobrazují na svítivých diodách úrovně na datové sběrnici a to tak, že při log. 1 se dioda rozsvítí a při log. 0 dioda zhasne. Při použití programového řešení se nám v pravém horním rohu zobrazí osmimístný displej, kde se zobrazí příslušný stav na datové sběrnici, a to přímo v 1 a 0.

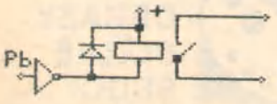
Dále si popišme praktické zapojení modulu. Toto zařízení můžeme realizovat v různém provedení a to podle dalšího využití:

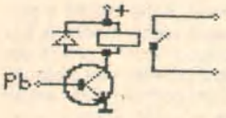
1. Nejjednoduší zapojení; pro každý bit datové sběrnice je použit jeden invertor typu 74LS04 a jedna LED dioda v následujícím zapojení:



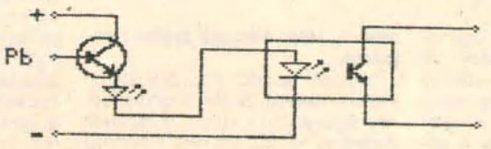
Toto se opakuje osmkrát pro celou datovou sběrnici a pro kontrolu ostatních pinů USERPORTu je možno přidělat další. Jako například signál STROBE, FLAG atd.

2. Složitější zapojení s možností kontroly a při současném ovládání dalšího zařízení lze vyřešit buď s připojením relé,





diodu LED můžeme zapojit do primárního obvodu, tím se rozumí u hradla nebo tranzistoru, nebo do sekundárního obvodu na straně kontaktu, nebo pomocí optočlenu



Optočlen má proti relé tu výhodu, že nemá příliš velké nároky na zdroj, to znamená, že lze bez potíží ze zdroje počítače napájet i větší počet optočlenů; u relé je to omezené. Další podstatný rozdíl je v hmotnosti a nároku na prostor na desce s plošnými spoji. Jinak splňuje oboji stejný úkol, to je oddělit počítač galvanicky od přídavného zařízení. Kontrolní LED diodu je opět možno připojit na obě dvě strany (jak je naznačeno ve schématu).

Nyní se budeme věnovat použití softwarového USERPORT displeje. Po přepsání programu, který je výše otištěn, si ho před spuštěním uložíme na paměťové médium (disk, kazeta), potom program spustíme; ten se nám uloží bokem v paměti (aby nevadil programům, které budeme dále používat) a v pravém horním rohu se objeví osm číslic, které znázorňují stav na jednotlivých Pb datové sběrnice. Nyní si naloudujeme program, který si potřebujeme vyzkoušet, a program spustíme. Na displeji můžeme kontrolovat stav datové sběrnice a to jak SOFTWARE tak HARDWARE displejem.

Po přezkoušení programů a funkcí zjistíme, že potřebujeme připojit více ovládaných prvků. To se dá řešit tím, že použijeme dekodér, a to buď 1 z 10 nebo 1 ze 16, a ty můžeme dál kombinovat podle potřeby.

Tímto jednoduchým způsobem můžeme rozšířit datovou sběrnici ze 4 na 16. Samozřejmě na zakódování obsluhy musíme pamatovat v programu.

JAN HORA

Počítačová gramatika

15) Píše ing. Rudolf Pecinovský, CSc.

Pokusme se nyní podle výše definovaného postupu vyřešit další příklad. Představte si Karla jako minéra, jehož úkolem je dojít před sebe k nejbližší zdi, tam položit další značku — minu a vrátit se zpět. Situace bude ztížena tím, že pokud Karel najde na nějakém políčku značky-miny, musí je nejprve odstranit, aby při přechodu dotyčného políčka nevybuchl. Na zpáteční cestě je pak opět musí položit všechny na přesně stejná místa, odkud je sebral, aby nikdo nemohl podle odjištěných min vypátrat jeho úkryt.

Začněme zase stejně, jako minule — najdeme si nejjednodušší případ, který budeme umět hravě vyřešit. Takovýmto případem bude opět situace, kdy bude Karel stát čelem u zdi. Položí pod sebe značku-minu, a je se vším hotov.

Složitější situace nastane, když Karel nebude u zdi. I zde však můžeme odhalit jednodušší a komplikovanější případ. V jednoduššim příkladě nebude pod Karlem žádná značka-mina a Karel bude moci úlohu vyřešit stejně, jako minule — tj. popojít o krok, vyřešit tuto jednodušší situaci a vrátit se zpět.

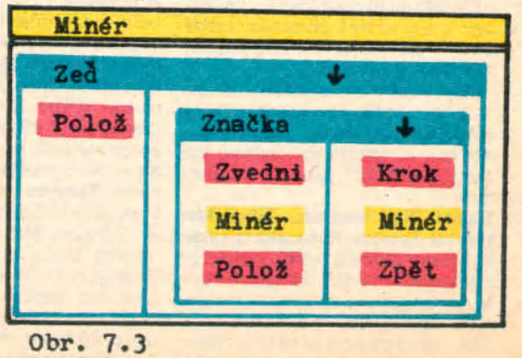
Ještě složitější situace nastane, když Karel nebude u zdi, ale budou pod ním navíc značky — miny. I zde budeme postupovat obdobně. Za nejjednodušší případ prohlásíme situaci, kdy je Karel sice na stejné pozici, ale není pod ním žádná značka — tedy případ, který jsme již vyřešili v předchozím odstavci. K tomuto nejjednoduššímu případu se přiblížíme tím, že Karel zvedne jednu ze značek, které jsou pod ním. A nopak — do původní výchozí pozice se dostane tak, že po splnění úkolu pod sebe jednu značku přidá.

8. OPAKOVÁNÍ

Tato kapitola uzavírá celý kurs programování, s nímž jste se setkávali déle než půl roku. Zopakujme si proto, co všechno jsme se v něm naučili. Pokud se vám bude zdát, že se v některých místech opakuji, věřte, že je to schválně, protože ne všem z vás se podařilo sehnat všechna potřebná čísla časopisu a tak si zde nejdůležitější myšlenky povíme ještě jednou.

Po seznámení s některými zásadami výuky programování, s nevýhodami dosavadních přístupů, a naopak výhodami přístupu použitého, a po seznámení s robotem Karlem a jeho světem jsme se hned na počátku vrhli na procedury.

Pověděli jsme si nejprve, jaký je rozdíl mezi primitivy a odvozenými příkazy a vysvětlili jsme si, že primitiva vlastně odpovídají strojovým instrukcím, resp. klíčovým slovům daného programovacího jazyka, a slova odvozená jsou ekvivalenty programů či podprogramů. Procedury a funkce jsou základními kameny celého moderního programování.



Obr. 7.3

Počítačová gramatika

16) Píše Ing. Rudolf Pecinovský, CSc.

Proto také v jazycích, v nichž nelze dobře vytvářet a používat procedury (např. klasický Basic) nikdo rozumný žádné složitější programy nepíše.

Při řešení ukázkových příkladů jsme si odvodili několik velice důležitých zásad:

1. Definici, která nevyhoví za jakýchkoli okolností nemůžeme považovat za správnou! Není tak důležité, jak je ta která činnost definována, tj. jakým postupem se dosáhne cíle. Důležitější než to, zda bude procedura pracovat o maličko rychleji, nebo že je o kousek kratší, je to, zda při plnění tohoto programu počítač za jakýchkoli okolností splní zadanou úlohu.

2. Chyby v definici nemusí být na první pohled patrné. Proto když vytvoříme nějaký program, a to nejen celý program, ale i jakoukoli jeho smysluplnou část, musíme se ihned přesvědčit, zda jsme jej nadefinovali správně. Projděte proto vždy po nadefinování procedury všechny speciální případy a ověřte, zda v nich nebude vaše procedura havarovat nebo zda se nebude chovat jiným nevhodným způsobem.

3. Bude-li se vám někdy zdát, že zadanou úlohu můžete řešit několika způsoby, vyberte si ten, který vede k řešení nejrychleji. V žádném případě se nesnažte hned na počátku vytvořit nejdokonalejší možný program. Většinou v něm uděláte tolik chyb, že se vám jej nakonec vůbec nepodaří odladit. Začněte proto nejprve nejjednodušší alternativou, která vás napadne, a teprve po jejím dokončení začněte program zdokonalovat.

4. Nehoňte se ve svých budoucích programech vždy za maximální efektivitou programu. Není zcela vyloučeno, že pokud napíšete program raději přehlednější, ušetříte tím čas nejen sobě, ale i budoucím uživatelům vašich programů.

Pojem procedur velice úzce souvisí s otázkou dekompozice, které byla věnována třetí kapitola. Zopakujem si ještě jednou, že základním strategickým heslem moderního programování je stará římská zásada „Rozděl a panuj". Cílem veškerého počátečního snažení každého programátora je rozdělit nejprve celý problém na rozumně velkou množinu rozumně malých a rozumně složitých problémů, a to tak, aby bylo možno jednotlivé problémy dále řešit samostatně. Řekli jsme si, že umění dekompozice složitých úloh na posloupnost úloh jednodušších je jedním ze základních pilířů umění programovat, a že s jistým zjednodušením lze dokonce říci, že kvalita programátora se pozná podle toho, jak dobře dokáže dekomponovat problém.

Jedním z klíčových kamenů dekompozice je správná volba mezimodulového rozhraní. Mezimodulové rozhraní definuje stav řešení problému v místech, kde si jednotlivé podprogramy předávají řízení, případně i některé detaily vlastního předání řízení. Jinými slovy, mezimodulová rozhraní definují pro každou proceduru stav, v němž program přebírá a stav, v němž jej předává.

I v této kapitole jsme si na závěr shrnuli řadu důležitých zásad:

1. Nepodlehněte počátečnímu zdání, že problém je naprosto triviální, a věnujte vždy alespoň chviličku jeho rozmyšlení nad papírem. Neuspěchávejte zasednutí za klávesnici počítače. Nepouštějte se zbrkle do vlastního programování a zamyslete se nejprve nad tím, zda by nebylo vhodné celou úlohu rozdělit na několik jednodušších podúloh.

2. Pokud je ze zadání zřejmé, že se bude některá část programu opakovat, bývá většinou výhodné naprogramovat tuto část jako samostatnou proceduru.

3. Úlohu je v zájmu zvýšení přehlednosti často vhodné rozdělit na podúlohy i v případě, že v ní žádné opakující se části nejsou.

4. Jednotlivé části, na něž úlohu rozdělujete, volte tak, aby tvořily logicky relativně samostatné a ucelené jednotky.

5. Nezapomeňte nejprve nadefinovat a v zápětí zkontrolovat mezimodulové rozhraní. Bez ujasněného mezimodulového rozhraní nepřistupujte k řešení jednotlivých podúloh.

6. Pokud je již dekompozice problému hotova a máte navržené a zkontrolované mezimodulové rozhraní, podívejte se, jestli některé z podúloh již nejsou natolik jednoduché, že je pro vás snadné je přímo naprogramovat. Pokud ano, naprogramujte je.

7. Ze zbylých podúloh vybírejte jednu po druhé a aplikujte na její řešení tyto body. Pořadí řešení není důležité. Někdo vybírá podúlohy od nejjednodušší k nejsložitější, jiný dodržuje pořadí jejich výskytu v programu, třetí volí vlastní, zcela specifický klíč výběru.

Ve čtvrté kapitole jsme se učili začleňovat do svých programů rozhodování. Ukázali jsme si, tři základní typy rozhodovacích bloků, a tedy i rozhodovacích příkazů, i když prvé dva byly vlastně speciálními případy třetího typu.

Pátá kapitola byla celá zasvěcena cyklům. Pověděli jsme si, že rozeznáváme čtyři základní druhy cyklů podle toho, která z obou podmínek (vstupní a výstupní) je pinohodnotná, a která degenerovaná (tj. není to podmínka v pravém slova smyslu, ale logická konstanta s hodnotou ANO nebo NE).

1. Nekonečný cyklus má obě podmínky degenerované. Když na něj narazíme v programu, vstupní podmínka nás vždy pustí dovnitř, ale výstupní podmínka nás nikdy nepustí ven. Pokud do něj jednou vstoupíme, nemůžeme se z něj dostat normální cestou ven. Metody, jak opustit nekonečný cyklus samozřejmě existují, ale nepatří do výzbroje úplného začátečníka, protože ten s nimi většinou natropí více škody, než užitku.

2. Cyklus se vstupní podmínkou má vstupní podmínku pinohodnotnou a výstupní podmínku degenerovanou (vždy nepravdivou). Vstupní podmínka se testuje před každým provedením těla cyklu. To znamená, že je-li vstupní podmínka hned na počátku nepravdivá, neprovede se tělo cyklu ani jednou!

3. Cyklus s výstupní podmínkou je cyklus s degenerovanou (vždy pravdivou) vstupní podmínkou a pinohodnotnou podmínkou výstupní. Výstupní podmínka se testuje až po provedení těla cyklu, a proto se tělo cyklu s výstupní podmínkou provede vždy alespoň jednou!

4. Cyklus s oběma podmínkami má, jak již jeho název napovídá, obě podmínky pinohodnotné. Tato varianta cyklu je již natolik rafinovaná, že v řadě programovacích jazyků ani nejde naprogramovat a proto se musí obcházet.

Šestá kapitola byla věnována funkcím a jejich použití v programech. Vysvětlili jsme si, co to znamená, když se řekne, že funkce vrací hodnotu, a nebo že funkce má vedlejší efekt.

Sedmou kapitolu jsme věnovali vrcholné programátorské lahůdce — rekurzi. Vysvětlili jsme si její základní princip na pohádce o kohoutkovi a slepičce a snažili jsme se vás přesvědčit, že před ní vůbec nemusíte mít takový strach, jak je to mezi programátory obvyklé.

Zopakujeme si ještě jednou jeden z typických rekurzivních postupů. Postup, pomocí nějž jsme vyřešili náš typový příklad.

1. Našli jsme nejjednodušší výchozí situaci, v níž jsme zadaný problém uměli vyřešit.

2. Našli jsme způsob, jak lze složitější situace převést na situace jednodušší, tj. na situace blíže situaci nejjednodušší, kterou již umíme řešit.

3. Zároveň jsme našli způsob, jak vyřešení jednodušších situací převést na vyřešení situace původní.

4. Celý problém jsme naprogramovali tak, že jsme otestovali, zda zadaná situace není nejjednodušší. Pokud ano, vyřešili jsme ji, pokud ne, převedli jsme ji na situaci jednodušší, na ni jsme aplikovali právě psaný program a výsledné řešení jsme převedli na řešení situace původní.

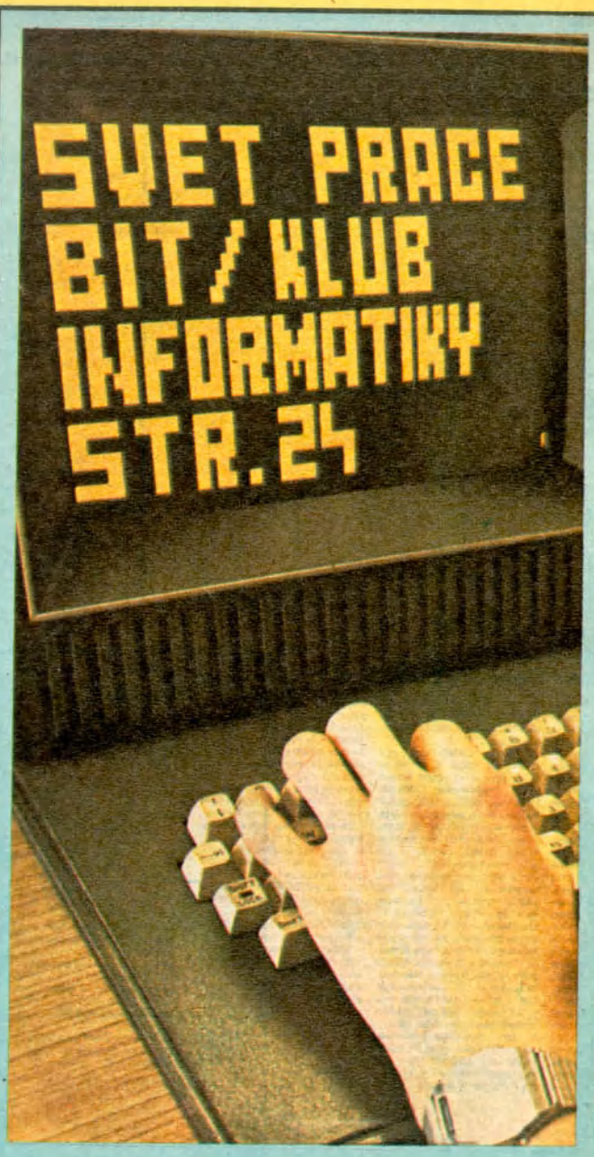
9. JAK DÁL?

Pokud vás programování během četby našeho kursu opravdu zaujalo a chtěli byste se mu věnovat i nadále, máte několik doporučeníhodných možností jak pokračovat.

Chcete-li i nadále zůstat samouky a nebo nemáte-li ani jinou možnost, mohu vám doporučit dva pokračovací kursy. Oba jsou věnovány programovacímu jazyku Pascal. Prvý vychází v časopisu Elektronika od 2. čísla ročníku 1988 a měl by pro vás tu výhodu, že přímo navazuje na tento kurs. Druhý vychází v časopise Věda a technika mládeži od 1. čísla ročníku 1988 a je výhodnější pro ty, kterým výuka prostřednictvím robota Karla nesedí a raději přijmou matematičtější způsob výkladu.

Daleko výhodnější je však nezůstávat samouky a přihlásit se do některého ze zájmových útvarů, které vznikly pod Svazarmem, Československou vědeckotechnickou společností či Socialistickým svazem mládeže, nebo při stanicích mladých techniků a klubech vědeckotechnické činnosti mládeže. Jak vidíte, organizací, v nichž se můžete výpočetní technikou zabývat, je dost. A mají pro vás další výhodu: všechny, s výjimkou ČSVTS, přijaly ve svých kroužcích a klubech jako nosnou metodiku Karla tak, jak jste se s ním seznámili v našem kursu a jak je dále používán v kursu v časopisu Elektronika.

KONEC



Za více než půldruhého roku existence bit/klubu SP, jak dokládá repro obálky č. 2 z roku 1987, jsme otiskli desítky článků o osobních počítačích, klubech a akcích týkajících se výpočetní techniky. Ve dvou seriálech jsme především laickým majitelům počítačů a zájemcům o výpočetní techniku nabídli možnost srozumitelnou formou se seznámit s obsahem terminů používaných v tomto oboru a připravit se tak na spolupráci s odborníky v klubech výpočetní techniky, kam kroky každého majitele počítače nutně musejí vést. Zvláště amatér se bez pomoci těchto kolektivů neobejde. Proto jsme se rozhodli seriál ukončit a zájemce odkázat na další, a nyní zcela odborné, zdroje informací. Z dopisů vysvítá, že mnohým z vás skutečně pomohl při pronikání do této zajímavé oblasti. Svědčí o tom i žádosti o zaslání chybějících čísel. Pokud jsme mohli, vyhověli jsme. Zásoby jsou však již téměř vyčerpané a těm, kteří „nezachytili nástup" nezbyde, než si vypůjčit a xeroxovat.

Seriál skončil, skončil i bit/klub na straně 24. V poněkud menším rozsahu jej však budete dále nacházet na dvoustraně 26—27.

Za všechny vaše dopisy děkujeme a i nadále očekáváme příspěvky do naší (i vaší) rubriky. Jistě mnohé z vás potěší citát z dopisu, který přišel jako na zavolanou: *Sestavil jsem program — programovací jazyk Karel k výuce programování napsaný v „čistém" Basicu a tedy jednoduše převoditelný pro různé typy mikropočítačů. Na uveřejnění v časopise je dlouhý, ale jsem ochoten jej nahrát (pro C16, C116, C+4) nebo upravit pro jiné počítače. Myslím, že by o něj mohl být zájem. Podepsán Pavel Machek, 588 12 Dobronín 100.* S autorem souhlasíme. Vhodná tečka za seriálem.