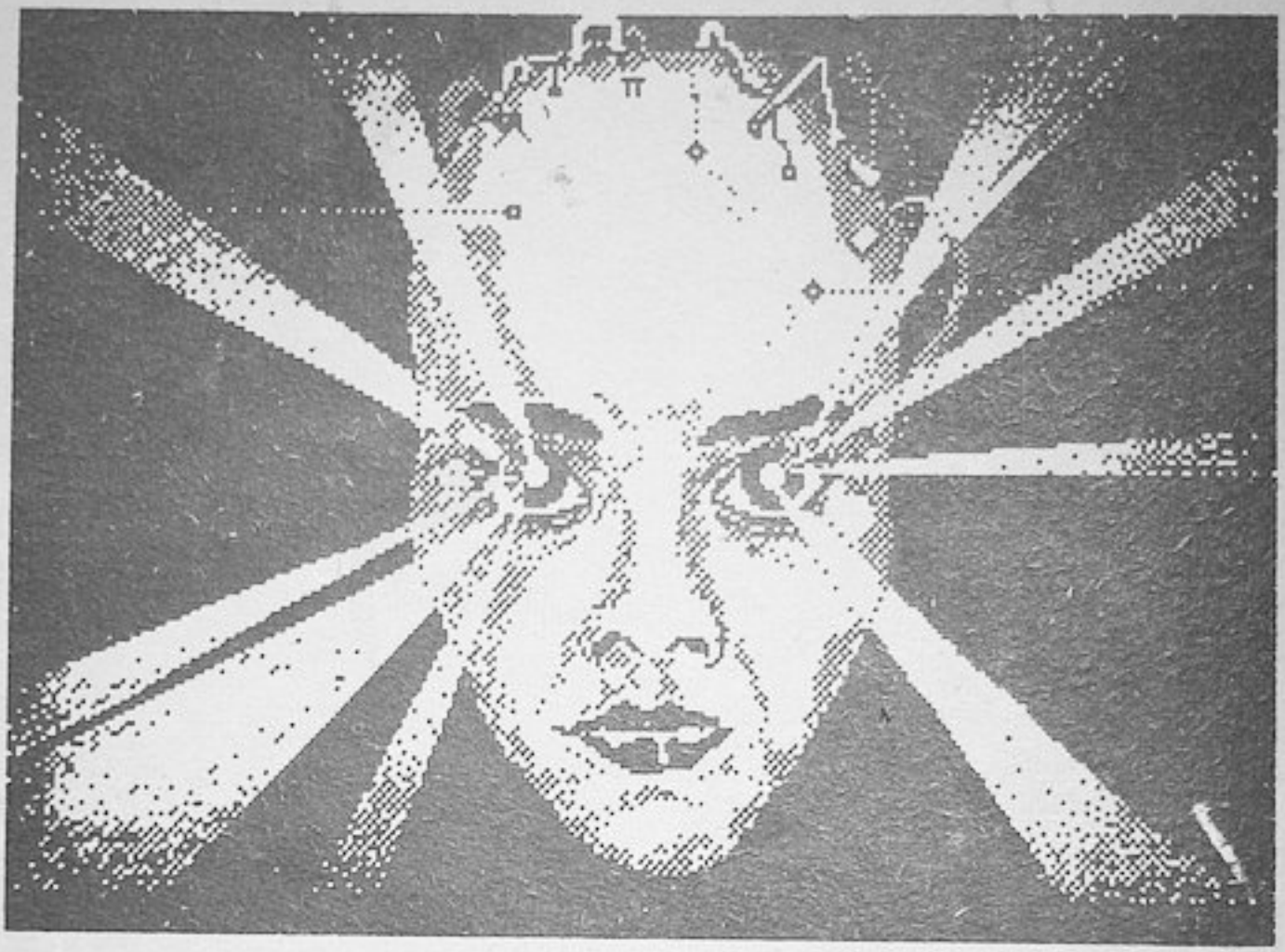


MŮJ PŘÍTEL

DIDAKTIK GAMA

ÚVODEM.

Tento "spis" vznikl z překladů zahraniční literatury a jeho účelem je pomoci všem, kteří si pořídili "domácího šotka", mikropočítač DIDAKTIK GAMA, SPECTRUM, nebo DELTA.

Vše podstatné, co potřebujete vědět do začátku, jste se již jistě dozvěděli z manuálu, ale zbývá ještě mnoho tajemna, ze kterého se zde pokusím alespoň růžek poodhalit. Zkušenější programoví borci mi budou snad vytýkat, že se zabývám samozřejmostmi, ale ruku na srdce - kolik z těch, kteří se s mikropočítači teprve seznamují, o těchto "samozřejmostech" vědí? Přitom by každý (nebo lépe téměř každý) se chtěl stát neomezeným vládcem té malé černé škatulky, kterou má doma.....

Doufám, že časem se připojí se svou troškou do mlýna i další autoři a podělí se o své znalosti s Vámi, kteří jen pozvolna vnikáte to tajemství hmoty zvané MIKROPOČÍTAČ.

Ještě bych rád připomenul, že pro stručnost všude používám označení SPECTRUM, ale vše se pochopitelně vztahuje i na všechny odvozené typy - od DIDAKTIKU GAMA až po Madarskou DELTU.

ČÍM ZAČNEME?

Domnívám se, že by bylo zbytečné opakovat to, co jste se již dozvěděli z manuálu, například o tvorbě znaků pomocí UDG. Proto půjdeme poněkud dále. Přitom se nebudeme zdržovat vysvětlováním základních pojmů, jako je systémová proměnná a podobně, komu by to nebylo jasné, at se vrátí k manuálu.

Začneme tedy tím, že si zde uvedeme jeden užitečný podprogram a několik šikovných definovaných funkcí, které budeme potřebovat k sestavování dalších programků.

Tak například v dalším budeme častěji potřebovat uložit pomocí POKE hodnotu do dvoubajtové proměnné, nebo tuto dvoubajtovou hodnotu číst atd. V prvém případě použijeme podprogram DPOKE, ve druhém definovanou funkci FN D a podobně. Takže začneme programem číslo nula - základním:

PROGRAM Ø

```
9890 REM DPOKE (adr,db)
9892 POKE adr,db-256*(INT (db/256))
9894 POKE adr+1,INT (db/256)
9896 RETURN
```

Tento program požaduje dva parametry: adr a db. Parametr adr je adresa nižšího bajtu, (t.j. bajtu umístěného v bunce paměti s nižší adresou), db je číslo v rozsahu od Ø do 65535, které chceme vložit do paměti na udanou adresu.

Dále si uvedeme několik definovaných funkcí, které budeme ve svých programech používat nejčastěji.

```
9900 DEF FN D (a)=PEEK a+256*PEEK (a+1)
9902 DEF FN T (w,n)=INT (w/2↑n)<>2*INT (w/2↑(n+1)
     )
9904 DEF FN A (w,k)=22528+32*w+k
9906 DEF FN P (w,k)=INT ((ATTR (w,k)-64*INT (ATTR
     (w,k)/64))/8)
9908 DEF FN W (c$)=8* CODE c$+FN D (23606)*( CODE
     c$<128)+(USR "a"-1152)*( CODE c$>=144)
```

NOVÉ ZNAKY.

Generátor znaků je sice umístěn pevně v ROM, ale v systémové proměnné CHARS je uvedena adresa jeho začátku. Tuto adresu (CHARS je v RAM) je již možno upravovat. Na začátek zkusme zvětšit hodnotu proměnné CHARS o 3. Nižší bajt (méně významný) je normálně Ø, zkusíme tedy:

```
POKE 23606,3
```

Od této chvíle jsou znaky ve své horní části pozměněny. Není v tom nic divného, vždyť nyní je pouze 5 bajtů znaku původních, další 3 bajty jsou převzaty z prvých 3 bajtů následujícího znaku. Nyní provedeme:

POKE 23606,8

Od této chvíle vypadají znaky opět normálně, ale místo "a" je vytištěno "b", místo "1" se objeví "2", místo mezery se objeví výkřičník (CODE " " = 32, CODE "!" = 33). Co se stalo? Zvýšili jsme počáteční adresu generátoru znaků o 8 bajtů - právě tolik zabírá sestava jednoho symbolu. Sestava znaku "mezera" předchází sestavu znaku "!", při tisku mezery pak počítač sáhne na adresu, kterou ukazuje CHARS, ale na této adrese je nyní místo mezery výkřičník - CHARS ukazuje o 8 bajtů výše než obvykle.

Samotný generátor znaků, který je umístěn v ROM tedy přemístit nejde, nanejvýš jsme počítač popletli tím, že jsme mu zadali chybnou adresu.

Možnost změnit adresu generátoru znaků ale můžeme využít jinak. Představme si, že v daném úseku paměti RAM vytvoříme nový generátor znaků. Samozřejmě, že musíme dodržet zásady jeho organizace: skupiny jsou osmibajtové a jejich pořadí odpovídá pořadí kodů určených znaků. Tvary znaků můžeme definovat úplně celé, podle vlastní úvahy. Méně používané znaky můžeme např. nahradit symboly, které se v základní sestavě nevyskytují, ale pro své účely je budeme potřebovat. Přeladíme tedy SPECTRUM na novou sestavu znaků tím, že do systémové proměnné CHARS vložíme novou adresu, zmenšenou o 256. Od této chvíle SPECTRUM "zapomene" na generátor znaků v ROM a píše na obrazovku pouze námi vytvořené znaky.

Přenos generátoru znaků do paměti RAM použijeme tehdy, kdy nám záleží na správném používání znaků české abecedy. Mohlo by se zdát, že by bylo jednodužší nadefinovat tyto znaky pomocí UDG. Ano, jenomže toto řešení je v praktickém použití nevýhodné. Připomenme si, že písmena á, é, ě, ř atd. se vyskytují dost často. Při vyvolávání znaku náležícího do sady UDG je zapotřebí nejprve přepnout do grafického modu, současným stiskem CAPS SHIFT a "9" a po napsání znaku se opět vrátit z grafického modu zpět, což je zdlouhavé, může docházet k omylům a značně zdržuje při psaní delších textů. Lepším řešením je vynechat některé z méně používaných symbolů z klávesnice a jejich přímou náhradou

často používanými znaky. Bez větších problémů je možno zrušit symboly "@", "$", "&", "£", atd. Každý z těchto znaků je snadno dostupný pomocí tlačítka SYMBOL SHIFT.

Umístění generátoru znaků v RAM se může hodit také tehdy, kdy počet znaků dostupných z UDG je nedostatečný, např. při programování her a ve výukových programech, ve kterých potřebujeme řecké písmena, chemické symboly, azbuku atd. Ve většině případů nebudeme tvořit všechny znaky od základu, ale zkopírujeme je z paměti ROM a pak provedeme příslušné úpravy .

Volba oblasti pro uložení nového generátoru znaků vyžaduje trochu přemýšlení, potřebujeme 96*8=768 bajtů. Vhodným místem tedy bude prostor pod UDG. Adresu začátku generátoru v tomto případě získáme: PRINT USR "a" - 768

V některých případech postačí deklarovat dva nebo více bloků UDG a v případě potřeby je přesunout zadáním nové hodnoty proměnné UDG. Pamatujme si, že proměnná UDG - na rozdíl od proměnné CHARS - obsahuje adresu prvého bajtu obsahu UDG, ne zmenšenou o 256.

Vyzkoušejme nyní své síly přenosem generátoru znaků do RAM, kam jsme si již určili, tedy hned pod UDG. Pochopitelně začneme rezervováním příslušného rozsahu paměti zabezpečeného před případným přepsáním programem v BASIC, zde si pomůžeme příkazem CLEAR. Následuje smyčka FOR - NEXT, která zkopíruje všechny bajty znaků z ROM do RAM. Takže do práce:

PROGRAM 1

```
10 CLEAR USR "a" - 769
20 LET noveznaky= USR "a" - 768
30 LET slareznaky= FN D (23606)+256
40 FOR i=0 TO 767
50 POKE noveznaky+i, PEEK (slareznaky+i)
60 NEXT i
70 LET db=noveznaky-256: LET adr=23606
80 GO SUB 9890: REM DPOKE
90 STOP
```

V programu je použit podprogram DPOKE, který jsme si uvedli hned na začátku (program 0).

Jako argument příkazu CLEAR vložíme adresu nového generátoru znaků, zmenšenou o 1. Argument CLEAR nyní ukazuje na poslední bajt přístupný jazyku BASIC, tento bajt se případně ještě může změnit.

Po překopírování znaků se pomocí podprogramu DPOKE počítač přepne na novou sestavu znaků, tím, že vloží novou hodnotu do systémové proměnné CHARS. Při zkoušce nepozorujeme zpočátku žádný rozdíl - vždyť nový i starý generátor znaků jsou naprosto shodné.

Předpokládejme, že požadujeme použití českých písmen. Tyto jsou všechny odvozeny od písmen, která jsou již ve SPECTRU obsažena. V prvé řadě překopírujeme počáteční tvar znaků, zde si pomůžeme funkcí FN W (viz řádek 9900 definovaných funkcí):

PROGRAM 2

```
100 INPUT "Znak k nahrade: ";a$
110 IF LEN a$=0 THEN STOP
120 INPUT "Novy vyznam znaku: ";b$
130 FOR i=0 TO 7
140 POKE FN W (a$)+i,PEEK (FN W (b$)+i)
150 NEXT i
160 GO TO 100
```

Program po spuštění si vyžádá zadání znaku, který nahradíme novým, čili méně potřebný znak, jako např. "%", kterému přiřadíme např. "á" atd. Pokud jako odpověd vložíme prázdný řetězec (stiskneme samotné tlačítko ENTER), program ukončí činnost a zastaví se. V opačném případě se zeptá na znak, který budeme používat. V uvedeném případě vložíme "a". Program přepíše 8 bajtů daného znaku a požádá o další znak k přenosu, vložíme postupně ostatní znaky.

Pak již můžeme přistoupit k úpravám. Přidáme-li písmenu "a", které je nyní vloženo místo znaku "X" nahoře dva body, obdržíme žádané písmeno "ä".

Provedme tedy:

```
POKE FN W ("X"), BIN 00000100
POKE FN W ("X")+1, BIN 00001000
```

Při zapisování těchto příkazů nás zaskočí, že po stisku tlačítka "X" se na obrazovce objeví písmeno "a". Nesahejme hned po DELETE, vždyť je to znak, který jsme sami zaměnili. Po každém dalším stisku klávesy "X" se již bude tisknout "ä".

Skutečně, pokus se nám povedl, ale to není příliš velký důvod k jásotu. Celé provedení je dosti primitivní. Pokud se zamýšlíme zabývat modelováním znaků častěji, je lépe použít nějaký jednoduchý pomocný editační program. Takovýto program by měl umožnit vytisknout libovolný znak k úpravám ve zvětšeném měřítku, umožnit jeho úpravy pomocí kurzoru umístěného na příslušné místo a upravený znak vložit do paměti.

Vytištění zvětšeného znaku nám nebude působit obtíže, viz program 3. Tento program zvětší 8 krát zvolený znak a vytiskne jej do středu obrazovky. Nejprve si však vyžádá vložení symbolu, určeného ke zvětšení:

PROGRAM 3

```
10 INPUT "Zvoleny symbol: ";z$
20 CLS: PRINT AT 21,0;z$
30 FOR y=0 TO 7
40 FOR x=0 TO 7
50 IF POINT (x,y)=1 THEN PRINT AT 15-y,12+x;"X";
60 NEXT X
70 NEXT y
```

Políčko (0,0) se skládá z bodů s grafickými souřadnicemi 0-7, stejně pro x jako pro y. Postupně analyzujeme body tohoto políčka zdola nahoru a zleva doprava. V případě výskytu zaplněného bodu zaplníme odpovídající pole ve čtverci vytvořeném

ve sloupcích 12 - 19 na řádcích 7 - 15. Každé pole tohoto většího čtverce odpovídá jednomu grafickému bodu ve zkoumaném poli (Ø,Ø).

Výraz 15-y,12+x provádí převod mezi souřadnicemi znaku a odpovídajícím polem ve zvětšeném čtverci. Znaménko "-" před y je použito z toho důvodu, že souřadnice y jsou číslovány zdola nahoru, čísla řádků zase zhora dolů. Místo písmene "X" na řádku 5Ø je možno použít libovolný grafický znak, např. plné políčko (CHR$ 143). Náš program pouze ukazuje, zda body jsou vybarveny, nevšímá si bodů s barvou pozadí (papíru).

PROGRAM 4

```
200 CLS
210 INPUT "Upravit znak: ";a$
220 LET a= FN W (a$)
230 FOR i=Ø TO 7
240 LET bajt= PEEK (a+i)
250 FOR b=Ø TO 7
260 LET barva=6-3* FN T (bajt,b)
270 PRINT AT 9+i,19-b; PAPER barva;" ";
280 NEXT b
290 NEXT i
```

Místo toho, abychom nejprve umístili znak na obrazovce a teprve pak jej analyzovali bod po bodu pomocí funkce POINT, jsme zde vybrali přímou cestu. Vypočítáme adresu prvého bajtu znakového vzoru a vkládáme jej do proměnné a. Dále po jednom bereme všech 8 bajtů vzoru a přiřadíme je proměnné bajt. Funkcí FN T (viz seznam funkcí v úvodu) pak analyzujeme bit po bitu a v závislosti na hodnotě každého bitu (Ø nebo 1) vybarvíme odpovídající pole ve zvětšené kresbě znaku.

Při výskytu nulového bitu funkce FN T (bajt,b=Ø) přiřadí proměnné barva hodnotu 6 (t.j. žlutá barva). Pokud bit = 1, FN T (bajt,b=1) přiřadí proměnné barva hodnotu 3 (magenta). Nakonec jsou v odpovídajícím poli čtverce 8*8 uprostřed obrazovky

vytištěny prázdné políčka se žlutou nebo purpurovou barvou.

Znak se vinou funkce FN T kreslí pomalu. Jak viděl, jazyk BASIC zde není dostatečně efektivním programovacím jazykem. Zapamatujme si to!

A nyní nejobtížnější úloha - editace znaku. Pomůžeme si pohyblivým kurzorem, ukazovátkem. Tento se musí pohybovat ve středovém čtverci po všech 64 políčkách. Po ukázání musí mít možnost rozsvítit (vybarvit) nebo zhasnout vybrané políčko. K přemísťování kurzoru je nejvýhodnější použít klávesy "5" až "8", u kterých jsou uvedeny šipky ve čtyřech potřebných směrech. Stiskem klávesy "1" bod (políčko) rozsvítíme, stiskem "Ø" jej zhasneme.

Zbývá otázka, jak znázornit ukazatel tak, aby byl viditelný v rozsvícených i ve zhasnutých políčkách. Nejjednodužší bude rozjasnit políčko s ukazatelem tím, že na něj vložíme atribut BRIGHT = 1. Stiskem mezery se editace ukončí.

Aktuální poloha ukazatele bude určena proměnnými x (sloupec) a y (řadek). Počáteční hodnota těchto proměnných ukazuje na levý horní roh čtverce.

PROGRAM 5

```
300 LET x=12: LET y=8
310 GO TO 340
320 IF x=xØ AND y=yØ THEH GO TO 360
330 POKE FN a (yØ,xØ), PEEK FN a (yØ,xØ)-64
340 POKE FN a (y,x), PEEK FN a (y,x) + 64
350 LET xØ=x: LET yØ=y
360 LET k$= INKEY$
370 IF k$="5" AND x>12 THEN LET x=x-1
380 IF k$="8" AND x<19 THEN LET x=x+1
390 IF k$="6" AND y<15 THEN LET y=y+1
400 IF k$="7" AND y>7 THEN LET y=y-1
410 IF k$="1" THEN POKE FN a (y,x), BIN Ø1Ø11ØØØ
420 IF k$="Ø" THEN POKE FN a (y,x), BIN Ø1110000
430 IF k$<>" " THEN GO TO 320
```

Pole s kurzorem má zůstat rozsvícené. Funkce FN a určí adresu bajtu atributů tohoto pole. Do aktuální hodnoty tohoto pole vložíme 64 (BIN Ø1ØØØØØØ), čímž se bit 6, který řídí jas, změní na 1 beze změny ostatních atributů tohoto pole (řádek 34Ø) Souřadnice rozjasněného pole jsou uloženy v proměnných xØ,yØ. Další děj bude záviset na manipulaci s klávesami.

Funkce INKEY$ sleduje stav klávesnice a znak stisknuté klávesy uloží do proměnné k$. Tato proměnná je pak porovnávána se všemi přípustnými řídicími znaky: 5 - 8, 1, Ø a mezera. V okamžiku shody program vykoná odpovídající děj. Na příklad, je-li stisknuta klávesa 5, t.j. šipka doleva a kurzor se nenachází u levého okraje čtverce (což hlídá podmínka x>12), pak se aktuální hodnota souřadnice kurzoru x zmenší o 1, což odpovídá posunu kurzoru o jedno pole doleva. Podobně se postupuje po stisku dalších řídicích kláves.

Hodnota 1 označuje rozsvícené pole, což znamená vepsat odpovídající atributy: INK libovolný, PAPER 3, BRIGHT 1 (v tomto poli je ukazatel), FLASH Ø. Analogicky při mazání pole (zhasnutí) klávesou Ø se vepíší atributy: PAPER 6 (barva prázdného pole), BRIGHT 1, FLASH Ø.

Po stisku SPACE program podle instrukcí na řádku 43Ø vypadne z editace znaku a přejde na vyšší číslo řádku (v tomto případě zůstane stát). V případech, kdy byla stisknuta jiná nebo žádná klávesa se program vrátí na řádek 32Ø a opět testuje klávesnici.

Zapamatujme si, že řídicími klávesami jsme pouze měnili hodnotu proměnných x a y. Neznamenalo to automatické přemísťování ukazatele na obrazovce, o to se musíme postarat sami. Nejprve se přesvědčíme, zda se ukazatel posunul. Rozpoznáme to podle nerovnosti aktuálních souřadnic s předešlými. V tomto případě provedeme dva úkony:

- obrátíme poslední stav pole, které ukazatel opouští (x0,y0). Odečtením 64 od hodnoty atributu vynulujeme bit BRIGHT bez vlivu na ostatní bity.
- Rozsvítíme pole s novou polohou ukazatele.

Hotový, upravený znak potřebujeme přenést z obrazovky do paměti znaků. Docílíme toho tím, že postupně analyzujeme jednotlivé řádky čtvercového pole na obrazovce a podle nich sestavíme bajty znaku:

PROGRAM 6

```
500 FOR i=1 TO 7
510 LET bajt=0
520 FOR b=0 TO 7
530 IF FN P (8+i,19-b)=3 THEN LET bajt=bajt + 2^b
540 NEXT b
550 POKE a+i,bajt
560 NEXT i
570 GO TO 200
```

Proměnná i představuje pořadové číslo řádku znaku a zároveň číslo bajtu, proměnná b je číslo bitu. Barvu papíru kontrolujeme funkcí FN P (viz seznam def. funkcí). Pokud není zelená (=3), nastavíme příslušný bit. Kompletní bajt pak zapíšeme do paměti znaků. Po odeslání všech osmi bajtů do paměti si program vyžádá další znak k úpravám.

Náš program dovoluje jak návrh znaků, které patří do hlavního generátoru (vždyť jsme jej přece přenesli do RAM), tak i znaků UDG. Jistým zlepšením by bylo např. spojení programů 2, 4, 5 a 6. Pracně zhotovenou sadu znaků nyní nahrajeme na kazetu.

V případě UDG příkazem:

```
SAVE "UDG" CODE USR "a",168
```

Obsah hlavního generátoru znaků uložíme:

```
SAVE "ZNAKY" CODE FN D (23606)+256,768
```

Jednou již vytvořené znaky je nyní možno použít v různých programech. Vložení UDG je jednoduché:

```
LOAD "UDG" CODE
```

Pokud ale má být obsah UDG umístěn jinam, než v době jeho vyhotovení (např. sestavený na SPECTRU 48, načítaný do SPECTRA 16), použijeme:

```
LOAD "UDG" CODE USR "a"
```

Nahrávání hlavního generátoru je o něco složitější, toto provádíme obvykle po opětném zapnutí počítače. Nejprve začneme oddělením potřebného rozsahu paměti pomocí instrukce CLEAR:

```
CLEAR USR "a"-769
```

a pak

```
LOAD "ZNAKY" CODE USR "a"-768
```

V případě, že generátor znaků ukládáme na totéž místo, ze kterého jsme jej nahráli na kazetu, pak postačí:

```
LOAD "ZNAKY" CODE
```

Po uložení nových znaků ještě musíme změnit hodnotu systémové proměnné CHARS:

```
LET adr=23606: LET db=USR "a"-256: GO SUB 9890
```

Nebo jednodušeji tak, že před nahrávkou generátoru znaků na kazetu přečteme oba bajty CHARS:

```
PRINT PEEK 23606, PEEK 23607
```

Údaje si zapíšeme, a po vložení generátoru do paměti je opět vložíme pomocí dvou příkazů POKE.

Generátor znaků je možno také umístit v jiném místě RAM, pouze je třeba se vyvarovat kolize s jazykem BASIC užitím příkazu CLEAR s příslušným argumentem.

PLAKÁTOVÉ PÍSMO.

Při psaní textů by se nám místo 32 znaků na řádku někdy spíše hodilo 42 nebo 64. Postačilo by definovat vlastní tvary znaků v poli 6*8 nebo 4*8 bodů a vložit program, který tyto znaky vykreslí na obrazovku. Tento program by ale v jazyku BASIC pracoval velmi pomalu. Občas by se nám hodilo i zvětšené písmo.

Vytvoříme si tedy univerzální podprogram, který by dovedl kreslit libovolné znaky tak, aby poloha znaku na obrazovce byla libovolně volitelná a aby bylo možno zvětšení nastavit nezávisle v obou osách.

Aby například bylo možno zvětšit znak v ose x dvakrát a v ose y třikrát, je třeba původní jeden bod znaku zaměnit za obdélník 2*3 body. Určíme si, že xØ a yØ označují souřadnice levého horního okraje pole znaku, zx a zy představují zvětšení v osách x a y, proměnná z$ bude obsahovat znak, určený k vytištění na obrazovce. Proměnné U a V budou postupně odečítat body určeného pole od Ø do 7.

Nejprve napíšeme podprogram, který na obrazovce vykreslí zaplněný obdélník o stranách délky zx a zy. Tento podprogram pak budeme používat místo příkazů PLOT k vytištění potřebných bodů znaku. Umístění obdélníku bude od xØ,yØ do U,V.

PROGRAM 7.

```
9870 REM pr(xØ,yØ,u,v,zx,zy)
9871 FOR i=Ø TO zx-1
9872 PLOT xØ+u*zx+i, yØ-v*zy
9873 DRAW Ø,1-zy
9874 NEXT i
9876 RETURN
```

Příkaz PLOT bude postupně rozsvěcet body horní strany obdélníka. Pokud zy>1, čili výška obdélníku je větší než jeden bod vykreslí příkaz DRAW směrem dolů úsečku o délce zy-1.

Organizace vlastní tvorby znaku již nepřináší žádné obtíže. Pomocí definované funkce FN W je určena adresa vzoru znaku. Nyní budeme postupně prohlížet jednotlivé bajty a jejich bity, podle nich pak budeme zaplňovat - nebo také ne - odpovídající pole sítě znaku:

PROGRAM 8

```
9860 REM zn(x0,y0,zx,zy,z$)
9861 LET w1= FN W (z$)
9862 FOR v=0 TO 7
9863 LET w2= PEEK (w1+v)
9864 FOR u=0 TO 7
9865 IF FN T (w2,7-u)=1 THEN GO SUB 9870
9866 NEXT u
9867 NEXT v
9868 RETURN
```

Nadešel čas zkoušení. Napíšeme příkazy, přiřazující hodnoty jednotlivým proměnným.

PROGRAM 9

```
10 INPUT "Znak: ";z$
20 LET x0=50: LET y0=160
30 LET zx=3: LET zy=5
40 GO SUB 9860
50 STOP
```

Pokud má být znak vytištěn inverzně, postačí v podmínce na řádku 9865 porovnávat hodnotu bitu (FN T) s nulou (0), místo s jedničkou (1).

Ačkoliv program pracuje, jeho rychlost nás nepotěší - kreslení jednoho znaku trvá minimálně několi sekund. Postačí ještě tak při tvorbě několika málo znaků, ale pro tvorbu více

znaků je celý postup příliš pomalý.

Pomalé provádění programů v jazyku BASIC jsme zjistili již dříve. Časové ztráty jsou způsobeny smyčkou, vykonávající velmi jednoduché operace. Radikálním řešením problému rychlosti je použití podprogramů napsaných v jazyku mikroprocesoru, ve strojovém kodu.

Proto se nyní vrhneme na:

ZÁKLADY STROJOVÉHO JAZYKA.

Programování ve strojovém jazyku není tajemná věda, vyhražená profesionálům. Základy tohoto umění zvládne každý, kdo dokáže logicky myslet, je trpělivý a systematický.

Úplný výklad strojového jazyka by zde byl příliš rozsáhlý. Znalost základů vnitřní řeči počítače nám však umožní pochopit jednoduché strojové programy a využívat je ve vlastních programech psaných v jazyku BASIC.

Strojový program je jediný, který mikroprocesor dokáže vykonávat přímo. Program v jazyku BASIC, zapsaný v paměti, je v podstatě symbolický. Jeho prováděním se zabývá rozsáhlý strojový program uložený v paměti ROM, tak zvaný interpreter jazyka BASIC. Tento program neustále analyzuje jeden znak po druhém, rozpoznává je a provádí v nich popsané činnosti v BASIC. Provedení jednoho příkazu pak vyžaduje od několika set do několika tisíc strojových příkazů. Většin těchto příkazů neslouží vlastnímu zpracovávání informace, ale spíše administrativním pracem – rozeznávání příkazů, kontrola jejich správnosti, vyhledávání proměnných v paměti atd. Právě z těchto důvodů (jsou ještě i další) se programy v BASIC vykonávají tak pomalu.

Mikroprocesor nepracuje s proměnnými podle jejich jména, ale s jednotlivými bunkami paměti s jejich vlastními adresami. Operace se provádí s bajty, v nejlepším případě s dvoubajtovými celými čísly.

Strojový program není souvislým čitelným textem, rozloženým do číslovaných řádků, ale řadou bajtů umístěných v určených bunkách paměti, převážně postupně za sebou. Při zápisu strojového programu, např. na papír, používáme obvykle t.zv. jazyk symbolických adres, zvaný také assembler, což slouží k srozumitelnění a "polidštění" programu. Před vložením do počítače je zapotřebí symbolický zápis převést na jemu odpovídající sled bajtů. V jazyku symbolických adres odpovídá jedné instrukci obvykle jeden strojový příkaz. Překlad je možno provést i ručně, ovšem u delších programů je takový postup pomalý a často pak může docházet k chybám.

V těchto případech používáme program, který překládá symbolický zápis do strojního kodu, nazýváme jej Assembler. Mezi nejznámější a nejpoužívanější programy tohoto typu patří např. GENS3, Dr.MG, MRS a další.

Mikroprocesor Z80 obsahuje několik jednobajtových nebo dvoubajtových buněk, nazývaných vnitřní registry, které se účastní ve většině operací. Strojové příkazy se provádí v uvedených registrech, takže zde nejsou potřebné takové adresy, jaké používáme při manipulaci s bunkami v paměti. Úplně nám postačí znát sedm jednobajtových registrů, označených písmeny A, B, C, D, E, H, L. Nejvíce možností má registr A, zvaný také Akumulátor. Registry B a C, D a E, stejně jako H a L je možno spojit do dvojic přechovávajících dvoubajtová čísla v celku.

Rozebereme si jednoduchý program, který se skládá pouze ze třech příkazů:

Strojový kod	Symbolický zápis	komentář
62	LD A, 255	; do registru A
255		; vlož konstantu 255D
50	LD (16385), A	; obsah registru přenes
1		; do bunky RAM s adresou
64		; 16385
201	RET	; vrať se do programu,
		; který vyvolal tento
		; podprogram.

Příkazy Z80 se skládají z 1 až 4 bajtů. První bajt je nejčastěji t.zv. operačním kodem, který označuje druh činnosti. Pokud jsou zapotřebí dodatečné informace, jsou obsaženy v jednom nebo dvou následujících bajtech.

Operační kod prvého příkazu je 62D. Mikroprocesor jej pochopí jako povel vzít bajt, který následuje za operačním kodem a vložit jeho hodnotu do registru A. Po provedení prvého příkazu si mikroprocesor přečte další bajt programu a znovu jej pochopí jako operační kod. Číslo 50D je podle vnitřního slovníku mikroprocesoru příkazem k přepsání (překopírování) obsahu registru A do bunky paměti s dále uvedenou adresou. Adresa pamětového místa (bunky) je dvoubajtová. Za operačním kodem je postupně uveden nižší a vyšší bajt adresy, v našem případě

$$1 + 256 * 64 = 16385$$

Je to adresa bunky v obrazové paměti.

V době provádění příkazu je hodnota A=255, tato hodnota tedy bude vložena do bunky číslo 16385.

Tyto dva popsané strojové příkazy fukčně odpovídají BASICovému příkazu POKE 16385,255 - jenže jejich provedení je mnohonásobně rychlejší.

U hloubavějších povah nyní může přijít otázka, zda by nebylo možné místo dvou zde uvedených příkazů použít jen jeden - tak tedy nikoliv. V repertoáru Z80 se nevyskytuje příkaz zápisu konstanty do bunky paměti s určenou adresou! Je to typická situace. Jednoduché a jasné příkazy, zapsané v BASIC, potřebují ve strojovém jazyku ještě více příkazů, někdy i několik set.

Poslední příkaz s operačním kodem 201D (t.j. RET) je jednobajtový a obsahuje pouze samotný operační kod. Zjednodušeně řečeno, tento příkaz umožnuje návrat ze strojového programu do BASIC (strojový program byl vyvolán z jazyka BASIC jako podprogram). RET zde plní podobnou úlohu, jako RETURN v BASIC.

Zapamatujme si, že v symbolickém jazyku Z80 nejprve uvádíme kam informace zaslat, a teprve potom, odkud je vzít.

Uvedený strojový program je přemístitelný (relokovatelný), to znamená, že může pracovat v libovolném místě paměti, na rozdíl od většiny strojových programů, které vyžadují umístění v pevně určeném místě paměti.

Nyní si tento prográmek uložíme do paměti počítače a vyzkoušíme jeho chod. Díky přemístitelnosti jej můžeme vložit do libovolného místa paměti, např. do buferu tiskárny (23296 - 23551). Ke vkládání použijeme příkaz POKE.

PROGRAM 10

```
10 FOR a=23296 TO 23301
20 READ x: POKE a,x
30 NEXT a
40 DATA 62,255,50,1,64,201
```

Uložení hodnoty jednotlivých bajtů do souboru DATA a jejich postupné čtení spolu s jejich zápisem do paměti ve smyčce FOR... ...NEXT je často používaný způsob vkládání krátkých strojových programů do paměti.

Ke spuštění strojového programu slouží ve SPECTRU funkce USR. Jejím argumentem je adresa, od které se má strojový program začít provádět. Poněkud zvlášlní způsob spouštění strojového programu vyvoláním funkce má svůj význam. Strojové programy často provádí operace, jejichž výsledky se mají předat do programu v jazyku BASIC. Funkce USR tento požadavek splnuje - v okamžiku vykonání příkazu RET obsahuje číslo v rozsahu 0 - 65535 které představuje obsah registrového páru BC.

Vymažme nyní obrazovku a zkusme:

```
LET x=USR 23296
```

Ve druhém sloupci nahoře na obrazovce se objeví černá čárka. Hodnota proměnné x nás nezajímá, příkaz přiřazení (LET) byl použit pouze pro volání funkce USR. Místo LET je také možno použít RANDOMIZE, pokud nepřekáží jeho vlastní činnost:

RANDOMIZE USR 23296

Strojový program, uložený v paměti, je možno upravovat. Například chceme vepsat místo 255 hodnotu 199, a ještě k tomu do bunky 16395, místo do 16385. V prográmku 10 by pak bylo zapsáno LD A, 199. Operační kod se samozřejmě nemění, upravíme tedy pouze druhý bajt - argument:

POKE 23297,199

Podobně i ve druhém příkazu. Druh operace se nemění, operační kod tedy opět zůstává, upravíme pouze nižší bajt adresy:

POKE 23299,11

Po spuštění pomocí RANDOMIZE USR 23296 vypíše program na obrazovku místo jedné čárky dvě krátké (199D = BIN 11000011) ve dvanáctém sloupci.

První prográmek byl jenom ukázka. Podíváme se ale na program o něco málo složitější, ale zato užitečný. Jeho úkolem bude zaplnit paměť atributů bajtem s udanou hodnotou, například 71D = BIN 01000111 (černý papír, bílý inkoust, zvýšený jas).

STROJ. KOD	SYMBOLICKÝ ZÁPIS	KOMENTÁŘ
33,0,88	LD HL, 22528	;adresa začátku paměti ;atributů do registr. ;páru HL
1,0,3	LD BC, 768	;počet bajtů paměti ;atributů do reg. ;páru BC
54,71	LD (HL), 71	;vyšli konstantu do ;bunky paměti s adresou ;uvedenou v HL
35	INC HL	;obsah reg. páru HL ;zvětši o 1
11	DEC BC	;obsah reg. páru BC ;zmenši o 1

```
120          LD A, B          ;obsah registru B
                              ;vlož do registru A
177          OR C             ;proved logický součet
                              ;registrů A a C
32, 248      JR NZ, $-8       ;není-li výsledek nula,
                              ;vrať se o 8 míst
201          RET              ;návrat do BASIC
```

Příkaz LD HL, 22528 si představuje registry H a L jako jeden šestnáctibitový registr. Údaj, který je do něj vepsán, je adresa prvé bunky bloku atributů (0+256*88=22528). Obdobně příkaz LD BC, 768 vepíše do registrů B a C, braných jako celek, hodnotu 768. Příkaz LD (HL), 71 [operační kod 54] umožnuje zapsat do vybrané bunky paměti konstantu, uvedenou ve druhém bajtu příkazu (v našem případě tedy 71). Adresa bunky paměti je uvedena obsahem registrového páru HL. Jelikož HL obsahuje v této chvíli číslo 22528, tak do bunky s touto adresou je vložena uvedená konstanta.

Jednobajtový příkaz INC HL zvětší o 1 (inkrementuje) číslo, uvedené v HL (původně bylo 22528, bude tedy 22529).

Příkaz DEC BC zmenší (dekrementuje) o 1 obsah BC (bylo 768, bude 767).

Příkaz LD A, B vloží do registru A hodnotu, obsaženou v registru B a příkaz OR C provede logický součet registrů C a A. Logický součet se provádí samostatně s každou dvojicí odpovídajících bitů. Výsledek operace bude nulový pouze tehdy, jestliže oba operandy (čísla účastnící se operace) budou také nulové. Jelikož jsme do registru A pokaždé vložili obsah registru B, příkaz OR C vlastně zjišťuje, kdy bude obsah registrového páru BC nulový.

Příkaz JR NZ, $-8 označuje podmíněný skok. Podmínkou provedení skoku je nenulový výsledek předešlé operace (JR NZ - Jump Not Zero - skoč, není-li nula). Skok se provede nazpět o 8 buněk paměti, čili na příkaz LD (HL), 71.

Příkaz k podmíněnému skoku bude platný tak dlouho, dokud obsah registru B se postupným odečítáním jedničky (dekrement) nezmenší na nulu. Tímto způsobem jsme ve strojovém programu vytvořili smyčku, která proběhne celkem 768krát. Po každém průběhu smyčkou se hodnota uložená v reg. páru HL zvětší o jednotku (inkrementuje), takže konstanta 71 bude vložena do 768 následných buněk paměti RAM, počínaje bunkou s adresou 22528.

V jazyku BASIC by bylo možno činnost tohoto programu napsat takto:

PROGRAM 11

```
10 LET HL=22528
20 LET BC=768
30 POKE HL, 71
40 LET HL=HL+1
50 LET BC=BC-1
60 IF BC <> 0 THEN GO TO 30
```

Proměnné HL a BC nemají samozřejmě nic společného s registry procesoru, jejich označení bylo zvoleno pouze pro názornost příkladu. Zkusíme tento program spustit, jeho průběh trvá kolem 12 sekund.

Nyní vyzkoušíme jeho protějšek ve strojovém kodu. Naštěstí i tento je relokovatelný, takže jej umístíme např. do bufferu tiskárny.

PROGRAM 12

```
100 FOR a=23296 TO 23310
110 READ x: POKE a,x
120 NEXT a
130 DATA 33,0,88,1,0,3,54,71,35
140 DATA 11,120,177,32,248,201
```

Necháme proběhnout zaváděcí program a příkazem RANDOMIZE USR 23296 spustíme strojový program. Vzhled obrazovky se nyní změnil v jediném okamžiku, provedení celého programu trvalo kolem 10 milisekund.

Náš program je univerzální, t.j. může být použit k zaplnění určeného obsahu paměti RAM libovolnou hodnotou. Postačí jenom změnit počáteční adresu, počet bajtů určených k zaplnění a hodnotu konstanty, která tyto bajty zaplní.

Zkusíme např. zaplnit celou obrazovou paměť bajtem 51D = BIN 00110011.
Nejprve zapíšeme nový argument příkazu LD HL, konstanta; pomůžeme si zde dříve uvedeným podprogramem DPOKE.

Provedeme tedy:

```
LET adr=23297: LET db=16384: GO SUB 9890
```

nebo můžeme také argument rozložit na nižší a vyšší bajt (méně a více významný):

```
POKE 23297,0: POKE 23298,64
```

Podobně postupujeme i s příkazem LD BC, konstanta. Hodnota konstanty nyní bude 6144:

```
POKE 23300,0: POKE 23301,24
```

Nakonec ještě vložíme novou konstantu, tvořící argument příkazu LD (HL), konstanta:

```
POKE 23303,51
```

Upravený program spustíme opět pomocí RANDOMIZE USR 23296, nyní získáme opět jiný výsledek než minule. Také v tomto případě se ale program vykonal bleskurychle.

HEXADECIMÁLNÍ ČÍSLA

Postupné vkládání bajtů programu do paměti ze souboru DATA je jednoduché, na to nám stačí obyčejná smyčka. Často se ale vyskytne potřeba zavést do paměti delší program, několik set nebo i více bajtů. Pak se zápis dat v desítkové soustavě ukáže být velmi neekonomický. Při přepisování programu, např. z časopisu, může také velmi snadno dojít k chybě, jejíž hledání ve spoustě dat spotřebuje mnoho času a hlavně nervů. Je tedy na čase zajímat se o lepší a ekonomičtější metodu vkládání strojových programů do operační paměti.

Místo zápisu v desítkové nebo dvojkové soustavě se používá šestnáctkové (hexadecimální) soustava, ve které je každý bajt zakodován do dvou hexadecimálních číslic. V soustavě o základu 16 potřebujeme 16 různých číslic. Sadu desítkových Ø - 9 tedy doplníme šesti prvými písmeny abecedy: A až F. Hodnoty těchto znaků jsou následující:

HEX.ČÍSLICE	DEKADICKÁ HODNOTA	DVOJKOVÁ HODNOTA
Ø	Ø	ØØØØ
1	1	ØØØ1
2	2	ØØ1Ø
3	3	ØØ11
4	4	Ø1ØØ
5	5	Ø1Ø1
6	6	Ø11Ø
7	7	Ø111
8	8	1ØØØ
9	9	1ØØ1
A	1Ø	1Ø1Ø
B	11	1Ø11
C	12	11ØØ
D	13	11Ø1
E	14	111Ø
F	15	1111

Přechod z dvojkové (binární) do šestnáctkové (hexadecimální) soustavy je jednoduchý. Postačí čtyřbitové skupině přiřadit odpovídající hexadecimální číslice. K zápisu jednoho bajtu v hexadecimální soustavě potřebujeme 2 znaky a právě tolik bajtů paměti programu v jazyku BASIC.

Před použitím ale musíme převést hexadecimální čísla na desítková (dekadická). Jedním z řešení je vytvoření souboru vzorových znaků, který obsahuje všechny hexadecimální čísla uložená postupně za sebou. Překládanou číslici v hexadecimálním vyjádření bychom pak porovnávali postupně s jednotlivými znaky tohoto řetězce. Po zjištění shodnosti, na základě čísla pořadí vzoru v řetězci, získáme hodnotu číslice:

PROGRAM 13

```
10 INPUT "Hexadecimalni cislo: ";h$
20 FOR i=0 TO 15
30 IF h$ <> "0123456789abcdef" (i+1) THEN NEXT i
40 PRINT "Dekadicka hodnota: ";i
```

Při postupném převádění stovek a tisíců hexadecimálních číslic nám bude vadit malá rychlost programu, kterou má na svědomí vícekrát probíhaná smyčka. Proto využijeme znalost kodu jednotlivých znaků:

PROGRAM 14

```
10 INPUT "Hexadecimalni cislo: ";h$
20 LET i= CODE h$-48-7*(h$>"A")
30 PRINT "Dekadická hodnota: ";i
40 GO TO 10
```

Doporučuji ještě na začátek programu vložit řádek:

```
5 POKE 23658,8
```

který nám přepne počítač do modu psaní velkých písmen.

Číslo 48 je kodem číslice "Ø", "1" má zase kod 49 atd až do "9" s kodem 57. Po odečtení čísla 48 od kodu číslice získáme odpovídající výsledek. Kod písmene "A" je 65, "B" - 66, "F" - 7Ø Pokud by byla číslice vyjádřena znakem "A" až "F", musíme tedy ještě odečíst 7.

Jak se zabezpečíme proti chybám při přepisování programů? Všechny strojové programy, uvedené v tělo příručce, jsou zapsány následujícím způsobem:

Skupina bajtů je uvedena jako řetězec hexadecimálních čísel po dvou číslicích na bajt. Všechny bajty, tvořící jednu skupinu jsou sečteny a zbytek z dělení tohoto součtu číslem 256 představuje t.zv. kontrolní součet, doplněný do řetězce jako poslední dva bajty.

Nyní si zkusíme představit, že při opisování omylem změníme hodnotu některé číslice, nebo zaměníme mezi sebou dva sousední znaky. Zaváděcí program čte postupně hodnoty bajtů a sečítá je podobně, jako to bylo provedeno při přípravě programu do tisku. Součet se vydělí 256 a zbytek po dělení je porovnán s posledním, t.j. kontrolním bajtem. Pokud se výsledek neshoduje s zde uvedeným číslem, určitě jsme někde udělali chybu. Zaváděcí program pak ohlásí číslo vadného řádku, což usnadní hledání chyby.

Před vlastním blokem bajtů se nachází dvě desítkové čísla. Prvé je adresa prvého bajtu v paměti, od kterého se bude program ukládat, druhé uvádí, od kterého řádku programu začíná blok dat (pro případ chyby).

Konec bloku dat program rozezná podle prázdného řetězce " " K zavedení strojového programu do paměti stačí jednoduše zadat příkaz RESTORE s argumentem, ukazujícím na prvý řádek bloku dat a spustit zaváděcí program příkazem GO SUB 9990.

Tímto postupem je možno uložit do různých částí paměti více samostatných strojových programů.

A nyní vlastní zaváděcí program:

PROGRAM 15

```
9990 REM LOADER STROJOVÝCH PROGRAMU
9991 READ a,n
9992 READ 1$: LET l=LEN 1$: LET s=0: LET k=2: LET
     n=n+1
9993 IF l=0 THEN RETURN
9994 LET w2=CODE 1$(k-1): LET w1=CODE 1$(k)
9995 LET c=w1-48-7*(w1>64)+16*(w2-48-7*(w2>64))
9996 IF k<l THEN POKE a,c: LET s=s+c: LET k=k+2:
     LET a=a+1: GO TO 9994
9997 IF s-256*INT (s/256)<>c THEN PRINT "Chyba na
     radku ";n: STOP
9998 GO TO 9992
```

UŽITEČNÉ STROJOVÉ PODPROGRAMY

A nyní nám zbývá náš zaváděcí program (angl. LOADER) nějak využít. Uvádím proto několik krátkých, ale často užitečných podprogramů ve strojovém kodu, kterými si můžete zpestřit a vylepšit své programy v BASIC. Všechny jsou uloženy v bloku DATA, jsou v hexadecimálním tvaru.

Všechny dále uvedené podprogramy jsou relokovatelné, t.zn. mohou být umístěny a používány bez úprav v libovolném místě paměti RAM. Prvý podprogram je zde volen tak, aby se uložil v paměti od adresy 64000, adresy všech ostatních jsou určeny tak, aby podprogramy spolu nekolidovaly. Před vkládáním těchto podprogramů je nutno nejprve nastavit hranici RAMTOP příkazem CLEAR 63999.

V případě, že by se vyskytla potřeba přemístit podprogramy na jiné místo paměti (např. u verze 16k, nebo při spolupráci s jinými strojovými programy), postačí upravit adresu počátku ukládání, uvedenou v prvém řádku každého podprogramu. Adresu prvého bajtu budeme označovat jako adrp.

Je celkem výhodné mít všechny tyto podprogramy spolu se zaváděcím programem uloženy na kazetě a v případě potřeby kteréhokoli z nich jej přihrát do vlastních programů pomocí příkazu MERGE.

1. Rotace obsahu obrazovky doprava (23 bajtů, adrp=64000)

Podprogram přesune celý obsah obrazovky (24 řádků) o jeden bod doprava. Atributy se nepřesunují, při změně o jeden bod to ani nemá smysl. Body, které na pravé straně opouští obrazovku se objeví opět z levé strany. Vícenásobné vyvolání podprogramu ve smyčce FOR.....NEXT umožní plynulý posuv obsahu obrazovky doprava. Pomocí příkazu POKE adrp+13,175 (např. POKE 64013,175) můžeme upravit podprogram tak, že body, které opustí obrazovku na pravé straně, se již vlevo neobjeví a obrazovka tedy zůstane prázdná. Návrat k původnímu stavu: POKE adrp+13,31.

Po příkazu POKE adrp+4,128 se přesouvají pouze horní dvě třetiny obrazu, POKE adrp+4,64 přesune pouze horní třetinu. POKE adrp+2,72: POKE adrp+4,64 ovládá pouze dolní třetinu obrazu.

Adresa spuštění: adrp

Příklad použití (dvojnásobný přesun celého obrazu):

```
FOR i=1 TO 512: RANDOMIZE USR 64000: NEXT i
```

Program 16

```
9600 REM Rotace obrazu doprava
9601 DATA 64000,9601
9602 DATA "2100400EC00620E5DD17"
9603 DATA "E1DD7E1F1FCB1E231096"
9604 DATA "FB0D20EFC9E0",""
```

2. Rotace obsahu obrazovky doleva (23 bajtů, adrp=64023)

Pracuje podobně jako předešlý podprogram, avšak obraz se při každém vyvolání posouvá o jeden bod doleva. Abychom odstranili body, které se opět vrací z pravé strany, zadáme příkaz POKE adrp+13,175. Původní stav obnoví POKE adrp+13,23.

Posun dolních 2/3 obrazu - POKE adrp+4,128
Posun dolní 1/3 obrazu - POKE adrp+4,64
Posun střední části obrazu - POKE adrp+2,79: POKE adrp+4,64
Posun horní části obrazu - POKE adrp+2,71: POKE adrp+4,64

Příklad použití (odsunout střední část obrazu):

```
POKE 64025,79: POKE 64027,64: FOR i=1 TO 256: RANDOMIZE USR 64023: NEXT i
```

PROGRAM 17

```
9610 REM Rotace obrazu doleva
9611 DATA 64023,9611
9612 DATA "21FF570EC00620E5DD2D"
9613 DATA "E1DD7EE117CB162B1050"
9614 DATA "FB0D20EFC9E0",""
```

3. Plynulý scroll obrazu nahoru (28 bajtů, adrp=64046)

Podprogram umožnuje posun obsahu obrazovky nahoru o jeden bod. Vícenásobné vyvolání vytvoří plynulý, pro oko příjemný scroll. Volá se na adrese adrp.

Příklad použití (scroll obrazu nahoru o 8 bodů, t.j. o jeden řádek):

```
FOR i=1 TO 8: RANDOMIZE USR 64046: NEXT i
```

PROGRAM 18

```
9620 REM Plynuly scroll obrazu
9621 DATA 64046,9621
9622 DATA "0100FF5104C578CDB110"
9623 DATA "22E506207E71121C2C76"
9624 DATA "10F9D1C178FEBF20E9D9"
9625 DATA "C9C9",""
```

4. Zrcadlové otočení obrazu (43 bajtů, adrp=64074)

Podprogram umožnuje zrcadlové převrácení obsahu obrazovky - body z levé strany obrazu budou zaměněny body z pravé strany obrazu atd.

Pomocí příkazu POKE adrp+28,0 upravíme podprogram tak, že pravá strana bude zrcadlovým obrazem levé strany, levá strana zůstane beze změny. Zpět do původního stavu se dostaneme pomocí POKE adrp+28,18. Program se spouští od adresy adrp.

Příklad použití (dvojnásobné otočení obrazu):

```
RANDOMIZE USR 64074: PAUSE 50: RANDOMIZE USR 64074
```

PROGRAM 19

```
9630 REM Zrcadlove otoceni
9631 DATA 64074,9631
9632 DATA "1100400D6D80E1021208E"
9633 DATA "0019E5C506082B7CFE76"
9634 DATA "581A30101FCB1610FBBD"
9635 DATA "1F12130D20ECC1D110FF"
9636 DATA "E0C946777818F1E7",""
```

5. Výměna obsahu obrazovky (19 bajtů, adrp=64117)

Na obrazovku se vejde pouze 22 až 24 řádků po 32 znaků. Často je však zapotřebí rychlý přístup k dalším informacím, jako příklad je možno uvést učební programy, kde se střídá text s vyobrazeními a vysvětlivkami. V případě potřeby musí mít uživatel možnost stiskem jednoho tlačítka vyvolat odpovídající text, nebo příklady řešení. Tyto situace se vyskytují i ve hrách, kdy si střídavě vyvoláváme např. mapu terénu a výhled z kabiny pilota.

Dále uvedený podprogram umožnuje řešit tento problém velice snadno. Postačí rezervovat v operační paměti úsek o délce 6912 bajtů, čili velikost obrazové paměti. Při každém svém vyvolání podprogram vzájemně zamění paměť obrazu s pomocnou pamětí. Takto je možné třeba připravit text vysvětlivek a uložit jej do pomocné paměti, s následným během vlastního programu. V okamžiku potřeby vyvoláme podprogram a na obrazovce se objeví blok vysvětlivek. Původní obsah obrazovky je nyní uschován v pomocné paměti a zpět na obrazovku se vrátí následným dalším vyvoláním podprogramu.

Pomocná paměť obrazu je umístěna od adresy 56000; proto je nutno před případným použitím tohoto podprogramu vložit příkaz CLEAR 55999. Pomocnou paměť je také možno podle potřeby uložit na jinou adresu. Adresa pomocné paměti je uložena v místech adrp+1, adrp+2 a může být změněna pomocí programu DPOKE (program č. 0).

POZOR! Při prvém vyvolání podprogramu se může stát, že plocha obrazovky zčerná. Je to normální, místo dosavadní obrazové paměti je vložen vynulovaný obsah pomocné paměti, stačí nyní použít příkaz CLS.

Příklad použití (zápis dvou obrazů a jejich záměny):

```
10 CLS: PRINT "TEST PROGRAMU"
20 RANDOMIZE USR 64117: CLS
30 CIRCLE 100,70,50: CIRCLE 150,80,40
40 RANDOMIZE USR 64117: PAUSE 75
50 GO TO 40
```

PROGRAM 20

```
9640 REM Vymena obrazu
9641 DATA 64117,9641
9642 DATA "21C0DA1100401A4E77EB"
9643 DATA "791223137AFE5B20F4A8"
9644 DATA "C9C9",""
```

6. Dodatková obrazova paměť (39 bajtů, adrp=64136)

Procedury, použité v tomto podprogramu, pracují s pomocnou obrazovou pamětí a vykonávají současně tři funkce. První procedura (vyvolá se na adrese adrp) kopíruje okamžitý obsah obrazovky do pomocné paměti a samotný obraz nechá v původním stavu. Druhá procedura pracuje právě opačně - kopíruje obsah pomocné paměti do obrazové, přičemž se obsah pomocné paměti nemění. Tato procedura se vyvolá od adresy adrp+3.

Procedury mohou být použity např. při tvorbě grafických kompozic na obrazovce, k doplnění výkresu mírami a podobně. Po každé úpravě můžeme obsah obrazovky uschovat v pomocné paměti a bez obav o zničení kresby provádět další experimenty s dokreslováním. V případě, že se nám kresba nepovede, můžeme obnovit původní vzhled obrazu jeho smazáním a vyvoláním druhé kopírovací procedury. Procedura pracuje i s atributy.

Třetí procedura umožnuje uschování obsahu obrazu a pomocné paměti, což je výhodné v těch případech, kdy zpracovávaná kresba používá stejné prvky, které je možno uschovat v pomocné paměti a v okamžiku potřeby je vkreslit do obsahu obrazovky. Atributy nejsou uchovávány, ale barvy na obrazovce zůstanou. Vyvolává se na adrese adrp+19.

Pomocná paměť pro všechny tři procedury začíná od adresy 56000, což je ale možné změnit stejně, jako v předešlém programu.

Příklad použití (zpracovávaná kresba je uschována v paměti, kreslí se nový obrázek a oba se spojí).

```
10 CLS: CIRCLE 110,70,60: CIRCLE 90,90,17
20 RANDOMIZE USR 64136: CLS
30 PRINT "DRUHA KRESBA": CIRCLE 100,80,73
40 PAUSE 50: RANDOMIZE USR 64155
```

PROGRAM 21

```
9650 REM Dodatkova pamet obrazu
9651 DATA 64136,9651
9652 DATA "AF18013711C0DA2100CB"
9653 DATA "403001EB01001BED8015"
9654 DATA "C911C0DA2100400118EE"
9655 DATA "001AB677132310F90D93"
9656 DATA "20F6C9DF",""
```

7. Inverze obrazu (16 bajtů, adrp=64175)

Podprogram provede inverzi obrazu - světlé body budou tmavé, tmavé se změní na světlé. Vyvolává se na adrese adrp.

Příklad použití (obraz bude 2 sekundy negativní):

```
RANDOMIZE USR 64175: PAUSE 100: RANDOMIZE USR 64175
```

PROGRAM 22

```
9660 REM Inverze obrazu
9661 DATA 64175,9661
9662 DATA "2100400118007E2F779E"
9663 DATA "2310FA0D20F7C91A",""
```

8. Kreslení znaků v libovolném zvětšení.

Poslední z této série podprogramů je určen ke kreslení znaků na libovolné místo obrazovky (s přesností na jeden bod) v libovolném zvětšení, určeném nezávisle pro obě osy X a Y.

Na rozdíl od předešlých podprogramů je zde vyžadováno vložení parametrů - kod znaku, souřadnice levého horního rohu jeho pole a zvětšení pro obě osy. Potřebné parametry se vkládají pomocí příkazu POKE. Kod znaku se vepíše do adrp+1, může být z rozsahu 32 - 127 (hlavní generátor znaků), nebo 144 - 164 (UDG).

Souřadnice X se uloží na adresu adrp+3, souřadnice Y na adrp+4, zvětšení v ose X do adrp+6 a zvětšení v ose Y do adrp+7. Pokud se některé parametry, jako např. zvětšení, nebo jedna ze souřadnic, v dalším volání nemění, není je pochopitelně nutno vkládat znova. Volá se na adrese adrp.

Příklad použití (vykreslení písmen A, B, C, D ve středu obrazovky. Zvětšení v ose X je zadáno 3, v ose Y je 6):

```
10 POKE 64197,3: POKE 64198,6: POKE 64195,90
20 FOR c=1 TO 4: POKE 64192,c+64
30 POKE 64194,50+30*c: RANDOMIZE USR 64191
40 NEXT c
```

PROGRAM 23

```
9670 REM Kresleni znaku
9671 DATA 64191,9671
9672 DATA "3E41Ø1644621Ø3Ø5E538"
9673 DATA "ED5B365CCB7F2806ED3F"
9674 DATA "5B7B5CD6906F26ØØ2956"
9675 DATA "292919E5DDE12A8D5C21"
9676 DATA "228F5CE1C5D9Ø6Ø8D973"
9677 DATA "54DD7EØØD9ØEØ8D95DD4"
9678 DATA "87F5C5D5E5DCE922E1C3"
9679 DATA "D1C1F1ØC1D2ØF1D9ØDA3"
9680 DATA "D92ØEAC1Ø5C5152ØDD8Ø"
9681 DATA "DD23D9Ø5D92ØD5C1C936"
9682 DATA ""
```

Pro zrychlení vkládání všech osmi uvedených podprogramů do paměti si pomůžeme řádkem:

```
RESTORE 9600: FOR i=1 TO 8: GO SUB 9990: NEXT i
```

Uvedené strojové podprogramy, mimo to, že jsou krátké a jednoduché, umožnují vytvořit příkazy, které jsou v BASIC nemožné, nehledě na vyšší rychlost jejich provádění.

Tolik na úvod do bližšího seznámení se strojovým jazykem, nástrojem, který dává programátoru prakticky neomezenou vládu nad počítačem. Pokud vás zaujaly jeho možnosti a chtěli by jste docílit více, než zde bylo popsáno, mohu vám již předem prozradit, že připravuji příručku věnovanou výhradně strojovému jazyku mikroprocesoru Z8Ø, tak, aby tomu porozuměl skutečně každý.

PROGRAMOVÉ DROBNOSTI

Náš mikropočítač má sice jednoduchou obsluhu, to však na druhé straně přináší mnoho slabých míst a nedostatků. Jejich odstranění je však jednoduché, snadno můžeme použít DIDAKTIK GAMA, příp. SPECTRUM s vlastnostmi jiných, mnohem dražších počítačů.

1. Akustická kontrola klávesnice.

Slyšitelnost akustického signálu při stisku klávesy je možno zlepšit, vložíme-li do systémové proměnné PIP na adrese 23609 hodnotu kolem 5 až 50 (nejvhodnější je nutno vyzkoušet, každému vyhovuje něco jiného). Tato systémová proměnná slouží k počítání cyklů signálu při každém stisku klávesy.

2. Zvětšení rychlosti kurzoru.

Po nabytí větší zručnosti v obsluze klávesnice se při editování může zdát rychlost přesouvání kurzoru malá. Čas, který uplyne mezi následujícími pohyby kurzoru, udávaný v padesátinách sekundy, je uložen v systémové proměnné REPPER na adrese 23562. Počáteční hodnota je 5, podle potřeby ji můžeme snížit na 3 - 2.

3. Změna kurzoru v příkazu INPUT.

Mnoho mikropočítačů používá v příkazu INPUT kurzor ve tvaru otazníku. DIDAKTIK GAMA však speciální znak nemá, použitý kurzor "L" není dostatečně zřetelným symbolem, že systém očekává vložení informace. Pomocí POKE 23617,236 je možno docílit toho, že kurzor v příkazu INPUT dostane tvar otazníku. Stává se, že např. po chybě při vkládání údajů se kurzor vrátí do původního stavu. Proto je vhodné umístit tento příkaz POKE samostatně před každým příkazem INPUT.

4. Výpis volné paměti.

Nejjednodušeji jej lze zařídit pomocí PRINT 65535 - USR 7962, tedy využitím strojového podprogramu z paměti ROM.

5. Zjednodušení zápisu dat na kazetu.

Systém nahrávání dat na mgf. pásek se nám u DIDAKTIKu GAMA nemusí zdát právě nejvhodnější (nemožnost nahrávky jednoduchých proměnných atd). Pokud postupně nahráváme několik bloků, pak před začátkem nahrávání každého z nich nás počítač vyzve ke spuštění magnetofonu a stisku některé klávesy.

Hlášení a čekání na stisk klávesy odstraníme příkazem POKE 23736,187 před každým příslušným příkazem SAVE. V praxi se obvykle hlášení před prvou nahrávkou ponechá, což nám umožní spustit magnetofon. Všechny další bloky se již nahrávají automaticky. Příklad:

```
4560 SAVE "Blok 1" DATA x()
4570 POKE 23736,187: SAVE "Blok 2" DATA y()
4580 POKE 23736,187: SAVE "Blok 3" DATA t$()
```

V případech, kdy potřebujeme nahrávat větší počet bloků, je uvedený způsob neekonomický. Před každou nahrávkou, i jen velice krátkou, je do magnetofonu vyslána samostatná hlavička, což zbytečně prodlužuje dobu zápisu i čtení. Nahrávka všech dat spolu s programem nepřichází v úvahu v případě dlouhých programů, zde je východiskem nahrát na kazetu pouze tabulku proměnných. Příkaz SAVE vyšle do magnetofonu celý obsah paměti od úseku označeného systémovou proměnnou PROG až do konce tabulky proměnných. Pokud na dobu nahrávání zaměníme hodnotu v PROG za hodnotu uloženou ve VARS (adresa počátku tabulky proměnných), pak se na kazetu nahrají pouze proměnné.

PROGRAM 24

```
9980 LET o=PEEK 23635: POKE 23635,PEEK 23627
9981 LET o1= PEEK 23636: POKE 23636, PEEK 23628
9982 SAVE "Nazev"
9983 POKE 23635,o: POKE 23636,o1
9985 STOP
```

Proměnné nahrajeme z pásky zpět do počítače příkazem MERGE "Název". Tento způsob nahrávání dat však není možné ověřovat pomocí VERIFY.

6. Simulace příkazu GET.

V některých případech vkládáme informace znak po znaku a požadujeme registrovat každý stisk klávesy. Mnoho mikropočítačů zde používá příkaz GET. U DIDAKTIKu GAMA si pomůžeme tak, že funkci INKEY$ umístíme do smyčky, např.:

```
300 LET a$=INKEY$: IF a$="" THEN GO TO 300
```

Takové řešení je ale nevýhodné, jelikož rychlost vkládání se dá nastavovat jen špatně. Jednodužší je použít příkaz PAUSE 0:

```
300 PAUSE 0: LET a$=INKEY$
```

Vykonávání příkazu PAUSE 0 končí v okamžiku stisku libovolné klávesy, jejíž odpovídající znak je přečten funkcí INKEY$.

7. INPUT v libovolném místě obrazu.

Většina mikropočítačů, od PP 01 až po např. ATARI, umožnuje provedení příkazu INPUT na libovolném místě obrazovky. I když systém INPUTu u GAMY považuji osobně za výhodnější (nemáme pak "přeplácanou" obrazovku), může se někomu zamlouvat prvý způsob. Znak, nebo znaky, vložené klávesnicí, pak zůstanou na určeném místě obrazovky. GAMA používá pro vkládání dat systémové řádky, které se po stisku ENTER vymažou, což může někdy působit i potíže, zvláště při postupném vkládání většího počtu dat za sebou.

Dále uvedený podprogram zpracovává řetězec znaků a může pracovat podobně jako PRINT na libovolném místě obrazu. Vkládané znaky na obrazovce zůstanou, je respektována i klávesa DELETE pro vymazání chybně vložených znaků. Vkládání řetězce ukončíme stejně jako u klasického INPUT klávesou ENTER. Po návratu z podprogramu obsahuje proměnná v$ vložený řetězec.

Pokud požadujeme vkládání číselných údajů, pomůžeme si funkcí VAL: LET v= VAL v$

V řádku 9828 je použit kod 12, který sleduje stisk klávesy DELETE, výsledkem je umazání posledního znaku z řetězce v$. Aby tento znak byl smazán i na obrazovce, je kurzor posunut doleva znakem CHR$ 8, v místě vymazaného znaku se vytikne mezera a ukazatel je opět posunut znakem CHR$ 8.

PROGRAM 25

```
9820 REM INPUT AT
9821 LET v$=""
9822 PAUSE 0: LET w$=INKEY$
9823 IF w$> CHR$ 164 THEN GO TO 9822
9824 IF w$<" " THEN GO TO 9827
9825 LET v$=v$+w$: PRINT w$;
9826 GO TO 9822
9827 IF w$=CHR$ 13 THEN PRINT : RETURN
9828 IF w$=CHR$ 12 THEN IF LEN v$>0 THEN
     LET v$=v$( TO LEN v$-1): PRINT CHR$ 8;
     " "; CHR$ 8;
9829 GO TO 9822
```

8. PRINT USING

Formát provádění výpisů čísel příkazem PRINT je málo pružný. Zvláště zarovnání formátu do levého okraje je značně nezvyklé. Při tisku sloupce čísel, např. v tabulkách, je požadováno , aby sobě odpovídající řády čísel byly umístěny pod sebou. Mnoho mikropočítačů disponuje příkazem PRINT USING, u kterého je možno definovat umístění vytištěných údajů.

U DIDAKTIKU GAMA a podobných typů lze tento příkaz nahradit definovanou funkcí:

```
9914 DEF FN u$(x,p,u)="                    "( TO p-
     (x=0)- LEN (STR$  INT  ABS  x)-u-1)+"-"( TO
     x<0)+ STR$ (INT ABS  x)+( STR$  (ABS  x-INT
     ABS x+1+.5*10↑-u))(2 TO 2+u)
```

Řetězec mezer, uvedený na počátku definice, obsahuje 16 znaků. Funkce má tři argumenty - prvým je číslo, které chceme vytisknout, druhý označuje celkovou délku pole, které použité číslo zabírá (max. 16) a třetí argument definuje počet číslic za desetinnou tečkou (max. 6).

Čísla jsou zarovnána podle pravého okraje pole určené šířky. Při tisku sloupců odpadá další uvádění modifikátoru TAB.

V dále uvedeném příkladu je vytištěna tabulka druhých a třetích mocnin, přičemž použité sloupce mají rozdílné formáty:

PROGRAM 26

```
10 FOR x=1 TO 3 STEP .1
20 PRINT FN u$ (x,6,1); FN u$ (x↑2,8,2);
   FN u$ (x↑3,12,4)
30 NEXT x
```

9. ON....GO TO, ON....GO SUB

Některé dialekty jazyka BASIC disponují tzv. počítanými skoky, přímými i do podprogramů. Instrukce skoku obsahuje více možných adres skoku, jejichž výběr se uskuteční v závislosti na hodnotách následujícího výrazu:

```
ON N GO TO 100,230,500
```

Pokud výsledkem předchozí operace má proměnná n přiřazenu hodnotu 1, příkaz vykoná skok na řádek 100. Při n=2 na řádek 230 a při 3 na řádek 500. V případě, že hodnota řídicí proměnné je menší než 1, nebo větší než počet uvedených adres skoků, bude příkaz skoku bud ignorován, nebo dá chybové hlášení. Jelikož GAMA nedisponuje příkazy ON...Go TO a ON...GO SUB, můžeme je nahradit použitím logických výrazů:

```
IF n>0 AND n<4 THEN GO TO (N=1)*100+(n=2)*230+(n=3)*500
```

V daném okamžiku může být pravdivý pouze jeden z logických výrazů a tím i pouze jedna adresa skoku.

10. ON ERROR GO TO

V mikropočítači GAMA a podobných typech má výskyt jakékoli chyby v průběhu programu za následek přerušení jeho běhu spojený s výpisem chybového hlášení. Někdy však nastanou situace, kdy výskyt chyby nemusí způsobit přerušení běhu programu, ale odpovídajícím způsobem upraví jeho reakci, jako např. při chybě v zadávání dat. Některé mikropočítače disponují v jazyku BASIC příkazem ON ERROR GO TO ..... , který způsobí skok na uvedenou řádku v případě výskytu jakékoli chyby.

Tento chybějící příkaz je možno u GAMY nahradit použitím podprogramu ve strojovém kodu:

PROGRAM 27

```
9690 REM ON ERROR GO TO
9691 DATA 23296,9691
9692 DATA "110A5B2A3D5C73237241"
9693 DATA "C901401F113F5BFD7E4F"
9694 DATA "003C121321425C7123B4"
9695 DATA "702336012301030DEDDE"
9696 DATA "B0E5FD3600FFFE0C20F1"
9697 DATA "04FD3601CC21761BE59B"
9698 DATA "2A3D5C3603233613C931"
9699 DATA ""
```

Tento podprogram není relokovatelný, ovšem díky jeho umístění v bufferu tiskárny (na adresách 23296 - 23362) nemůže kolidovat s dalšími programy. Podprogram se aktivuje příkazem RANDOMIZE USR 23296, od tohoto okamžiku po jakékoliv chybě dojde ke skoku na programový řádek 8000. Na tomto řádku je pak umístěn začátek programu obsluhy chyby, v jazyku BASIC.

Číslo chyby (shodné s vnitřním číslováním chyb systému, viz manuál) je možno zjistit na adrese 23359. Číslo řádku, na kterém došlo k chybě, je uloženo na adresách 23360 a 23361, pořadové číslo příkazu v řádku je uloženo na adrese 23362, takže program obsluhy chyby disponuje plnou informací o místu výskytu i druhu chyby.

Po každé chybě je zapotřebí podprogram znovu aktivovat pomocí RANDOMIZE USR 23296, vystoupil ze systému se dá příkazem RANDOMIZE USR 23350. Pokud požadujeme, aby řízení programu po chybě bylo převedeno na jiný řádek, než 8000, vložíme číslo příslušného řádku na adresy 23307 a 23308.

Výhody, které plynou z tohoto podprogramu, názorně ukáže následující příklad:

PROGRAM 28

```
  10 RANDOMIZE USR 23296
  20 LET a=x
  30 LET w=30/0
  40 LET y=200: PLOT 30,y
  50 NEXT y
  60 LET x=1: LET y=1: READ p
  70 STOP
8000 LET cischyby= PEEK 23359
8010 PRINT "CHYBA ";cischyby; TAB 32
8020 LET radek= PEEK 23360+256* PEEK 23361
8030 PRINT "Na radku ";radek;"/ "; PEEK 23362
8040 PAUSE 100: IF cischyby=9 THEN STOP
8050 RANDOMIZE USR 23296: GO TO radek+10
```

V tomto příkladu je naprogramována celá řada chyb. Protože podprogram ON ERROR GO TO je aktivován, dojde po každé chybě skok na řádek 8000, kde budou vypsány údaje o čísle chyby a proměnné radek bude přiřazena hodnota, odpovídající číslu řádku

```
9660 REM Inverze obrazu
9661 DATA 64175,9661
9662 DATA "2100400118007E2F779E"
9663 DATA "2310FA0D20F7C91A",""
```

s chybou. Po vypsání čísla řádku a příkazu, ve kterém došlo k chybě, a po kontrole zda kod chyby není 9 (STOP), následuje opětná aktivace podprogramu ON ERROR GO TO a skok na další řádek za tím, ve kterém se naposled vyskytla chyba.

Závěrem ještě připomínka - pro vkládání čísla řádku, na který má podprogram skočit po chybě, je výhodné použít již známý podprogram DPOKE (PROGRAM Ø).

11. "Kulatější" kružnice.

Algoritmus vykreslení kružnice příkazem CIRCLE není u počítače DIDAKTIK GAMA právě nejdokonalejší, vadí zvláště špatná symetrie, rovněž rychlost kreslení není právě největší. Tyto důsledky vyplývají z faktu, že GAMA počítá průběh kružnice z celých hodnot proměnných, což při zaokrouhlování funkcí INT způsobuje chyby. Východiskem je použití jiného algoritmu kreslení kružnice. V těch případech, kdy rychlost není právě nejdůležitější, postačí i dále uvedený podprogram v BASIC:

PROGRAM 29

```
9810 REM Vylepsena kruznice
9811 LET x=Ø: LET da=INT (r/2)
9812 IF x>r THEN RETURN
9813 IF da<Ø THEN LET da=da+r: LET r=r-1
9814 LET da=da-x-1
9815 PLOT xØ+x,yØ+r: PLOT xØ-x,yØ+r
9816 PLOT xØ+x,yØ-r: PLOT xØ-x,yØ-r
9817 PLOT xØ+r,yØ+x: PLOT xØ-r,yØ+x
9818 PLOT xØ+r,yØ-x: PLOT xØ-r,yØ-x
9819 LET x=x+1: GO TO 9812
```

Před vyvoláním podprogramu přiřadíme proměnným xØ a yØ hodnoty, odpovídající souřadnicím středu kružnice, proměnné r její poloměr.

Tento podprogram vykreslí rovnoměrně všechny čtvrtiny kružnice, díky čemuž má ideální symetrii. Věnujme pozornost tomu, že výpočet jednotlivých bodů se děje bez použití trigonometrických funkcí, pouze s nejjednodužšími aritmetickými operacemi! Díky tomu probíhá kreslení relativně rychle i v jazyku BASIC.

Pro ty, kteří mají v oblibě rychlé efekty, je dále uvedena strojová verze tohoto programu.

PROGRAM 30

```
9700 REM SUPER CIRCLE
9701 DATA 23363,9701
9702 DATA "000000ED5B445BED4B1F"
9703 DATA "435BAF5F7A1F677BBAE1"
9704 DATA "D07CA7F25D5B82153D71"
9705 DATA "93F5D5C5CD6B5BC1D147"
9706 DATA "E11C18E8C5CD735BC11E"
9707 DATA "7B5A57C579834FCD7E87"
9708 DATA "5BC179934FC57882477D"
9709 DATA "CD895BC1789247CDAA3A"
9710 DATA "223C473E010F10FDB6B6"
9711 DATA "77C940",""
```

Také tento program není relokovatelný. Je rovněž uložen v bufferu tiskárny, tak aby nekolidoval s programem ON ERROR GO TO, na adresách 23363 až 23445. Spouští se od adresy 23366, na adresu 23363 vložíme souřadnici x a na 23364 souřadnici y středu kružnice. Poloměr uložíme na adresu 23365.

Tento podprogram nejenom že vykreslí značně dokonalejší kružnici než příkaz CIRCLE, ale hlavně je neporovnatelně rychlejší. Snadno se o tom přesvědčíme pomocí následujícího příkladu:

```
10 POKE 23363,128: POKE 23364,87
20 FOR r=1 TO 87 STEP 3
30 POKE 23365,r: RANDOMIZE USR 23366
40 NEXT r
```

Předpokládám, že před vložením tohoto programu do počítače již v něm máte zaveden strojový podprogram!

Uvedený příklad by nás měl vyléčit z přesvědčení o absolutní dokonalosti našeho počítače.

12. Kreslení úseček při použití absolutních souřadnic.

Příkaz DRAW využívá pro kreslení úseček zadané souřadnice. Často nás však zajímá provedení čáry od posledního použitého bodu do konkrétního bodu na obrazovce. V tomto případě je výhodné využít systémové proměnné COORDX (23677) a COORDY (23678), které průběžně obsahují poslední použité souřadnice bodu PLOT nebo DRAW.

Proměnné x a y použijeme jako souřadnice koncového bodu:

```
DRAW x-PEEK 23677,y-PEEK 23678
```

Využití systémových proměnných nám umožní zbavit se dodatečných proměnných a má tu výhodu, že zarovnává chyby zaokrouhlování, které se mohou akumulovat.

13. Kreslení přerušovaných čar.

Příkaz DRAW kreslí pouze nepřerušované, souvislé čáry. Při ilustracích, kreslení výkresů apod. však často potřebujeme čáru přerušovanou. Nejjednodušším východiskem je použití vlastního podprogramu, který kreslí úsečky podobně jako DRAW, ale dovoluje volný výběr druhu kreslené čáry

PROGRAM 31

```
9790 REM SUPERDRAW (dx,dy,dl)
9791 LET x0=PEEK 23677: LET y0=PEEK 23678
9792 LET w1=dl: IF dx=0 THEN GO TO 9797
```

```
9793 LET w=SGN dx
9794 FOR u=x0 TO x0+dx STEP w
9795 LET v=INT (dy/dx*(u-x0)+.5)+y0
9796 GO SUB 9802: NEXT u: RETURN
9797 IF dy=0 THEN RETURN
9798 LET w= SGN dy
9799 FOR v=y0 TO y0+dy STEP w
9800 LET u=INT ((v-y0)*dx/dy+.5)+x0
9801 GO SUB 9802: NEXT v: RETURN
9802 IF w1>=0 THEN PLOT u,v
9803 LET w1=w1+1: IF w1>=d1 THEN LET w1=-d1
9804 RETURN
```

Podprogram SUPERDRAW vyžaduje tři parametry, které by měly být definovány v programu, ze kterého se SUPERDRAW volá. Proměnné dx a dy jsou souřadnice konce úsečky, podobně, jako u příkazu DRAW. Proměnná d1 obsahuje charakter kreslené čáry. Pokud d1=0, je kreslena nepřerušovaná čára, při d1>0 je přerušovaná čára složená z úseček o délce d1. Počátek úsečky je opět stejný, jako u příkazu DRAW.

14. Skok na určitý příkaz v řádku.

Příkazy GO TO a GO SUB vykonávají skok na počátek řadku, jehož číslo je uvedeno jako argument příkazu. Může se ale, např. při ladění programu, vyskytnout potřeba skoku ne na prvý, ale na některý další příkaz v řádku. Pak si můžeme pomoci jednoduchým podprogramem

PROGRAM 32

```
1000 REM SPEC-GOTO
1001 POKE 23618,radek-256*INT (radek/256)
1002 POKE 23620,prikaz
```

kde proměnná radek obsahuje číslo řádku a proměnná prikaz číslo příkazu na tomto řádku, na který se má převést řízení programu. Příklad:

```
LET radek=7500: LET prikaz=3: GO TO 1000
```

a řízení programu je převedeno na třetí příkaz v řádku 7500.

15. Totéž po LOAD.

Předešlý postup skoku na určený příkaz v daném řádku můžeme využít i při automatickém spouštění programu po nahrávce z pásku. Do programu vložíme např.:

```
POKE 60000,70: POKE 60001,0: POKE 60002,5
```

a na kazetu jej nahrajeme příkazem

```
SAVE "nazev" CODE 6E4,3
```

Zpět do počítače jej nahrajeme LOAD "nazev" CODE 23618,3. Program se automaticky spustí od příkazu 5 v řádku 70.

16. RENUMBER.

Po zdárném sestavení programu se občas může vyskytnout potřeba přečíslovat všechny programové řádky, "aby to lépe vypadalo". Přečíslovávat delší program ručně je poněkud namáhavé a zdlouhavé, proč tuto činnost nesvěřit počítači?

Použijeme tedy následující podprogram, který je ale maximálně zjednodušen, takže příkazy skoků GO TO a GO SUB musíme nakonec přečíslovat ručně, ale těch již není tolik.

PROGRAM 33

```
9900 REM Line Renumber
9901 PRINT "Prikazy GO TO a GO SUB je nutno
     precislovat dodatecne rycne!"
9902 INPUT "Cislo prveho radku: ";p
9903 INPUT "Krok cislovani: ";k
9904 LET n=PEEK 23635+256*PEEK 23636
9905 POKE n,INT (p/256): POKE n+1,p-256*INT
     (p/256)
```

```
9906 LET n=n+PEEK (n+2)+256*PEEK (n+3)+4
9907 IF 256*PEEK n+PEEK (n+1)=9900 THEN
     GO TO 9909
9908 LET p=p+k: GO TO 9905
9909 PRINT "Posledni radek je ";p
9910 PAUSE 0: CLS: LIST
```

17. Dvojitá velikost písma.

Tento podprogram se nám může hodit např. pro psaní nadpisů a zdůraznění některých částí vypisovaného textu. Souřadnice začátku nápisu jsou podprogramu předávány v proměnných x a y, text k výpisu v řetězci a$. Příklad:

50 LET x=0: LET y=0: LET a$="NADPIS TABULKY": GO SUB 9920

Písmena A a B na řádku 9927 jsou vložena v grafickém modu.

PROGRAM 34

```
9920 REM VELKE PISMO
9921 FOR a=1 TO LEN a$
9922 LET c=15616+(8*( CODE a$(a)-32))
9923 FOR b=0 TO 7
9924 LET d=( USR "a" + (b*2))
9925 POKE d,PEEK (c+b): POKE (d+1), PEEK (c+b)
9926 NEXT b
9927 PRINT AT y,x;"A";AT y+1,x;"B"
9928 LET x=x+1
9929 NEXT a: RETURN
```

18. Uložení obsahu obrazovky do řetězcové proměnné.

Pokud nechceme použít pro uložení obrazů program 20 nebo 21, můžeme obsah obrazovky (např. vysvětlující text) uložit do řetězcové proměnné a nechat si jej kdykoli vyvolat CLS: PRINT s$ Podprogram můžeme použít případně i jako doplněk programu 20, pokud potřebujeme uschovat více obrazů.

Tímto způsobem ale nejde uložit obrázek vytvořený příkazy DRAW, PLOT atd. Výhodou však je, že můžeme použít i více proměnných pro uložení více textů (např. v učebních programech) a kdykoli si kterýkoli z nich postupně vyvolat na obrazovku.

PROGRAM 35

```
9930 REM USCHOVA TEXTU
9931 DIM s$(704)
9932 FOR l=1 TO 21: FOR m=1 TO 32
9933 LET s$(m+32*l)= SCREEN$ (l,m)
9934 NEXT m: NEXT l
```

19. Funkce LEFT$, MID$, RIGHT$

Mikropočítače mnoha typů používají pro dělení řetězcových proměnných funkce LEFT$, MID$, RIGHT$. U našeno miláčka DIDAKTIK GAMA je vše zjednodušeno univerzálním kouzelným slůvkem TO, ale mohou se vyskytnout problémy (zvláště u méně zkušených, kterým je tato publikace určena) při přepisu programů z jiných počítačů, kde jsou tyto funkce použity.

Naštěstí máme možnost použít definované funkce, takže si všechny podobné případy můžeme snadno nasimulovat:

```
LEFT$ (A$,D)  -> DEF FN L$ (a$,d)=a$( TO d)
MID$ (A$,P,D) -> DEF FN M$ (a$,p,d)=a$(p TO p+d-1)
RIGHT$ (A$,D) -> DEF FN R$ (a$,d)=a$( LEN a$-d+1 TO )
```

Pak již můžeme klidně např. místo MID$ (A$,P,D) vyvolat novou funkci FN M$ (a$,p,d) a výsledek je stejný.

20. Převody úhlů.

Když jsme se již dostali ke vtipnému použití definovaných funkcí, uvedeme si ještě další příklady.

Tak třeba vzájemné převody úhlů ve stupních, radianech a gradianech můžeme snadno provést pomocí definované funkce:

```
DEF FN X (x,y)=x*((y=1)*PI/180+(y=2)*180/PI+(y=3)*10/9+
(y=4)*9/10+(y=5)*200/PI+(y=6)*PI/200
```

Vstupními hodnotami jsou: x - úhel, který chceme převádět
y - druh převodu podle tabulky:

```
1 = stupně   -> radiany
2 = radiany  -> stupně
3 = stupně   -> gradiany
4 = gradiany -> stupně
5 = radiany  -> gradiany
6 = gradiany -> radiany
```

21. Funkce ROUND (x,y)

Tato funkce je u některých dražších a komfortněji vybavených mikropočítačů použita k zaokrouhlování na předem daný počet desetinných míst (např. ve finančnických výpočtech na haléře atd). My sice máme mnohem levnější "stroj", ale tohle dokáže také pomocí definované funkce:

```
DEF FN X (x,y)= INT (x*10↑y+.5)/10↑y
```

kde proměnná x je zaokrouhlované číslo a y udává počet desetinných míst v rozsahu -8 až 8.

22. Funkce TRANSLATE (x$,y$,z$)

Používá se pro výměnu znaků v řetězcových proměnných, zde si pomůžeme krátkým podprogramem, který v případě potřeby vyvoláme příkazem GO SUB:

PROGRAM 36

```
9935 REM TRANSLATE (x$,y$,z$)
9936 FOR i=1 TO LEN x$: FOR j=1 TO LEN z$
9937 IF x$(i)=z$(j) THEN LET x$(i)=y$(j)
9938 NEXT j: NEXT i: RETURN
```

Program očekává tyto proměnné:

x$ - řetězec, ve kterém se má provést výměna
y$ - řetězec znaků sloužících pro výměnu
z$ - řetězec znaků, které se mají nahradit

23. Funkce MOD (x,y)

Opět funkce z dražších počítačů, výstupem je zbytek po dělení (modulo).

```
DEF FN X (x,y)=(x/y-INT (x/y))*y
```

kde x je dělenec a y dělitel.

24. Funkce INDEX (x$,y$)

Tato funkce vyhledává v dané řetězcové proměnné určený řetězec znaků.

PROGRAM 37

```
9940 REM INDEX (x$,y$)
9941 LET y=0: FOR i=1 TO (LEN x$ - LEN y$ + 1)
9942 IF x(i TO i + LEN  y$-1)<>y$  THEN  NEXT  i:
     RETURN
9943 LET y=i: RETURN
```

Vstupní údaje: x$ - řetězec, který se má prohledávat
y$ - řetězec s hledanou hodnotou

Výstup je v proměnné y - adresa bajtu, na kterém se oba řetězce překrývají, nebo 0, pokud hledaný řetězec znaků nebyl nalezen.

25. Funkce MAX (x,y)

Hodnotou této funkce je větší z obou čísel x,y

PROGRAM 38

```
10 REM MAX (x,y)
20 DEF FN M (x,y)=x+(y-x)*(y>x): REM maximum
30 DEF FN N (x,y)=x+(y-x)*(y<x): REM minimum
40 DEF FN T (x,y,z)= FN M (x, FN M (y,z)):
   REM pro tři parametry ald.
```

26. Převod HH.MMSS do dekadického tvaru

Slouží k převodu hodin-minut-sekund na číslo v desítkové soustavě. Vstup je proměnná x, což je argument pro převod z šedesátkové do desítkové soustavy.

```
DEF FN X (x)=INT  x+INT  ((x-INT  x)*100)/60+((x-INT  x)*100-INT
((x-INT x)*100))/36
```

27. Převod dekadického čísla na HH.MMSS

Je to funkce inverzní k předcházející - číslo z desítkové soustavy převede do šedesátkové, na tvar hodiny-minuty-sekundy. Vstupní proměnná y je opět převáděné číslo v desítkové soustavě.

```
DEF FN Y (y)=INT  y+INT  ((y-INT  y)*60)/100+((y-INT  y)*100-INT
((y-INT y)*36))/100
```

28. Formátování tisku čísel.

Program je podobný dříve uvedenému PRINT USING, avšak má více možností formátování. Vstupní číslo n vytiskne na řádek r, přičemž desetinná tečka je na t pozici od levého okraje obrazovky. Proměnné n,r,t jsou tedy definovány v hlavním programu, podprogram se volá GO SUB 9945 (čísla řádků je možno pochopitelně upravit dle potřeby). Používá se při tisku výsledků do tabulek a předem připravených formulářů.

PROGRAM 39

```
9945 REM TISK CISEL
9946 IF ABS n<.1 AND n<>0 THEN LeT t=t+1
9947 IF n<0 THEN LET t=t-1
9948 PRINT AT r,t-LEN STR$ INT ABS VAL STR$ n;n
9949 RETURN
```

TEXTOVÝ EDITOR

Počítač je dobrým nástrojem k redigování textů za podmínky, že je příslušně naprogramován. Redigování souvislých řádek textu a jejich následující skládání je nejjednodužší, ale nevýhodné. Mnohem lépe je operovat přímo na potřebném úseku textu, zabírajícím např. plochu celé obrazovky. Nastavíme kurzor na místo, ve kterém chceme provést jakoukoli změnu a přímo vpisujeme znaky do zvoleného místa obrazovky, mažeme je, nebo je nahrazujeme jinými. Po ukončení editace (redigování) obsah obrazovky jednoduše odešleme do paměti.

Mnoho počítačů, např. COMMODORE 64 disponuje tzv. obrazovkovým editorem, programem, umožnujícím upravovat a "tvarovat" texty na obrazovce přímo, interaktivně. GAMA tuto možnost nemá, ale můžeme ji vytvořit vložením příslušného programu v BASIC.

Minimální verze tohoto programu musí umožnovat volné přemístování kurzoru po obrazovce a vpisování znaků do vybraného místa. Vítaná by také byla reakce na klávesu ENTER, která umožní přejít na nový odstavec.

Dále uvedený program je vlastně příkladem takového jednoduchého obrazovkového editoru.

PROGRAM 40

```
1000 REM JENODUCHY TEXTOVY EDITOR
1010 LET x=0: LET y=0: LET x0=0: Let y0=1
1020 LET t$="": BORDER 6: GO TO 1130
1030 PAUSE 0: LET k$=INKEY$: LET k=CODE k$
1040 IF k<31 OR k>127 THEN GO TO 1060
1050 PRINT AT y,x;k$: LET k=9
1060 IF k=8 AND x>0 THEN LET x=x-1
1070 IF k=9 AND x<31 THEN LET x=x+1
1080 IF k=10 AND y<21 THEN LET y=y+1
1090 IF k=11 AND y>0 THEN LET y=y-1
1100 IF k=13 AND y<21 THEN LET x=0: LET y=y+1
1110 IF k=15 THEN GO TO 1200
1120 IF x0=x AND y0=y THEN GO TO 1140
1130 PRINT AT y0,x0; OVER 1; PAPER 7;" ";
1140 PRINT AT y,x; OVER 1; PAPER 5;" "o
1150 LET x0=x: LET y0=y: GO TO 1030
1200 FOR y=21 TO 0 STEP -1
1210 FOR x=31 TO 0 STEP -1
1220 LET t$= SCREEN$ (y,x)+t$
1230 NEXT x
1240 NEXT y
```

Proměnné x a y představují aktuální, x0 a y0 předešlou polohu kurzoru. Celý program je vlastně smyčka, která očekává znaky vkládané z klávesnice a příslušně na ně reaguje.

Na obrazovce je vidět kurzor, který tvoří pole vyplněné tmavším odstínem (PAPER 5). Kurzor se dá přemisťovat pomocí kláves 5 - 6 - 7 - 8, stisknutými současně s klávesou CAPS SHIFT případně u počítače SPECTRUM + samostatnými řídicími klávesami.

Kody těchto kláves (8 až 11) zjišťují řádky 1060 až 1090 a působí příslušnou změnu polohy kurzoru o 1 políčko. Řádek 1100 reaguje na stisk klávesy ENTER. Pokud se kurzor právě nenachází

na spodním okraji uživatelského okénka, přejde na následující odstavec, vynuluje souřadnici kurzoru x a souřadnici y zvýší o jednotku.

Pohyb kurzoru je prováděn pomocí příkazů na řádcích 1120 až 1150. Pokud se poloha kurzoru nezměnila, následuje návrat zpět na začátek programové smyčky. V opačném případě opustí původní políčko, na kterém se v tomto okamžiku nachází (1130) a následuje změna pozadí z bílého na modré (PAPER 5) v jeho novém postavení (1140). Nakonec jsou souřadnice kurzoru uloženy do proměnných x0 a y0.

Příkaz PRINT OVER 1;" "; nemá žádný vliv na obsah políčka, jelikož obraz mezery neobsahuje ani jeden zaplněný bod, který by mohl změnit stav některého ze zobrazených bodů, dojde puze ke změně atributů. Na řádku 1110 se vyhodnocuje stisk klávesy GRAPHICS (CAPS SHIFT a "9") zakončující editaci, provede se přenos obsahu obrazovky do dosud prázdné proměnné t$. Do ní jsou současně doplněny znaky načtené z uživatelského okénka funkcí SCREEN$.

Proměnná t$ však neobsahuje žádné informace o struktuře řádků, ale zůstanou uloženy všechny nepotřebné mezery, umístěné napravo od posledního znaku v každém řádku. Upravil by se to dalo změnou způsobu členi obrazovky (řádky 1210 - 1230):

PROGRAM 41

```
1210 LET w=0: LET t$=CHR$ 13+t$
1211 FOR x=31 TO 0 STEP -1
1212 LET k$=SCREEN$ (y,x)
1213 IF k$=" " AND w=0 THEN GO TO 1230
1220 LET w=1: LET t$=k$+t$
1230 NEXT x
```

Roli kontrolora zde plní pomocná proměnná w, která je před analyzou každého řádku vynulována. Na začátku je do pomocné proměnné t$ vložen znak ENTER (kod 13), který označuje konec

řádku, a pak jsou počínaje od konce do ní vkládány všechny další znaky. Proměnná w=0 označuje, že se dosud nevyskytl žádný znak, odlišný od mezery. Pokud při w=0 zjistíme mezeru, můžeme ji klidně ignorovat.

Výskyt jiného znaku než mezera způsobí jeho přepis do proměnné t$ a změnu hodnoty w z nuly na 1. Od této chvíle každá další mezera již určitě má vliv na strukturu řádku, a musí tedy být uložena do t$. Připomenme si, že uvedený postup je možný pouze při čtení od konce řádku!

Po proběhnutí programu neobsahuje proměnná t$ zbytečné mezery na konci řádků, místo nich zde máme znak ENTER. Takový text nyní můžeme nechat vytisknout na tiskárně a bude vytištěn ve stejném stavu, jaký je vidět na obrazovce.

Náš textový minieditor nelze považovat za dostatečně komfortní, při složitějších korekturách je nutno přepisovat celé dlouhé úseky. Mimo jiné zde chybí funkce mazání jednotlivých znaků textu s jejich současným nahrazováním úseky textu umístěnými vpravo od nich, hodila by se i možnost likvidace celých řádků, nebo rozhrnutí textu pro vložení dodatečných řádků.

Ve zlepšené verzi textového editoru použijeme pomocnou tabulku textu t$(32*22), která bude plnit funkci kopie obrazovky. Každému políčku obrazovky pak odpovídá jedno políčko tabulky t$.

Při každé operaci na obrazovce, např. vepsání znaku, provede se zároveň odpovídající úprava tabulky, tak, aby v každém okamžiku souhlasila s obsahem obrazovky.

Obsah tabulky t$ se na obrazovce objeví po stisku klávesy DELETE (kod 12). Nejprve upravíme ten úsek t$, který odpovídá textu od vymazaného znaku do konce řádku. Úsek je přesouván doleva o jednu pozici, na konci je doplněna mezera, poté je upravený řádek vytištěn celý znova.

K podobné operaci dojde po EDIT (kod 7). Zde je text od kurzoru do konce řádku přesunut doprava a na pozici kurzoru se objeví prázdné místo, do kterého je možné doplnit chybějící znak

PROGRAM 42

```
1000 REM TEXTOVY EDITOR
1010 DIM t$ (32*22)
1020 LET x=0: LET y=0: LET x0=0: LET y0=1
1030 BORDER 6:POKE 23562,2:GO TO 2000
1040 PAUSE 0: LET k$=INKEY$: LET k=CODE k$
1050 IF k<32 OR k>127 THEN GO TO 1080
1060 PRINT AT y,x;k$: LET t$ (1+x+32*y)=k$
1070 IF x<31 THEN LET x=x+1: GO TO 2000
1080 IF k=8 AND x>0 THEN LET x=x-1
1090 IF k=9 AND x<31 THEN LET x=x+1
1100 IF k=10 AND y<21 THEN LET y=y+1
1110 IF k=11 AND y>0 THEN LET y=y-1
1120 IF k=13 AND y<21 THEN LET x=0: LET y=y+1
1130 iF k=12 THEN GO TO 2100
1140 IF k=4 THEN GO TO 2200
1150 IF k=5 THEN GO TO 2300
1160 IF k=7 THEN GO TO 2400
1170 IF k=15 THEN POKE 23562,5: RETURN
2000 IF x=x0 AND y=y0 THEN GO TO 2020
2010 PRINT AT y0,x0; OVER 1; PAPER 7;" ";
2020 PRINT AT y,x; OVER 1; PAPER 5;" ";
2030 LET x0=x: LET y0=y: GO TO 1040
2100 IF x=0 THEN GO TO 1040
2110 LET p$=t$(x+32*y+1 TO 32*y+32)
2120 LET t$(x+32*y TO 32*y+32)=p$+" "
2130 PRINT AT y,0;t$ (32*y+1 TO 32*y+32);
2140 IF x>0 THEN LET x=x-1: GO TO 2020
2200 LET t$(1+32*y TO 704)=t$(33+32*y TO 704)
2210 PRINT AT 0,0; t$;: GO TO 2020
2300 LET p$="                                "
2310 LET t$(1+32*y TO )=p$+t$(1+32*y TO )
2320 GO TO 2210
2400 LET p$=t$ (x+32*y+1 TO 32*y+31)
2410 LET t$(x+32*y+1 TO 32*y+32)=" "+p$
2420 GO TO 2210
```

Ještě jednodužší je postup v případě vymazání řádku (klávesa TRUE VIDEO, kod 4). Celá operace se skládá z přepisu úseku t$ od začátku řádku do bodu, ve kterém je současná poloha kurzoru. Zbývající do 32 znaků budou automaticky nahrazeny mezerami, jak je to obvyklé při přepisu na řetězcích dané délky a celá změna t$ bude vytištěna na obrazovce.

Po stisku INVERSE VIDEO (kod 5) text, počínající od řádky s kurzorem, je přesunut o jednu řádku dolů a v uvolněném místě se objeví prázdný řádek, do kterého je možno vepsat chybějící text.

POZOR! Řetězec na řádku 2300 musí obsahovat 32 mezer!!

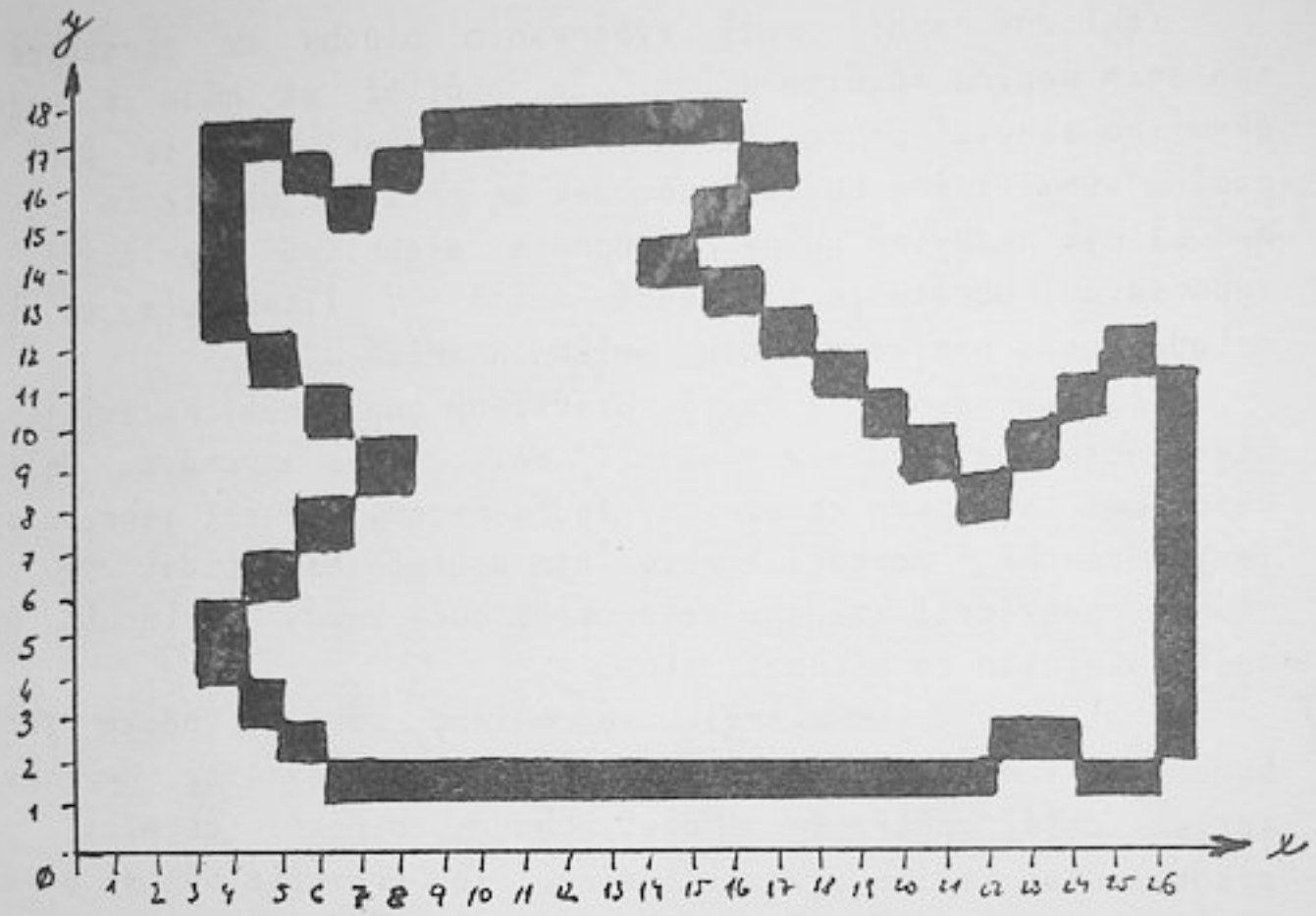
Klávesa GRAPHICS (kod 15) zakončí editaci. Proměnná t$ odstraní přebývající mezery a nahradí je znakem ENTER.

Také zde, podobně jako v předešlém programu (41) je možno provést dodatečné úpravy.

PAINT - VYBARVENÍ UVNITŘ OBRYSU

Častým požadavkem počítačové grafiky je vybarvení plochy, ohraničené uzavřeným obrysem. Uvažme, že obrys může mít libovolný, často značně členitý tvar, představující např. kresbu kontinentu. Dodejme ještě, že pro vybarvení plochy s libovolným obrysem postačuje zadat jeden libovolný bod uvnitř uzavřené plochy.

Na rozdíl od matematické geometrie operuje geometrie obrazovky s popsanými rozměry a umístěním. Za uzavřený obrys považujeme takový uzavřený řetězec bodů, ve kterém se jednotlivé body souvisle dotýkají libovolným způsobem - ať již stranou, nebo rohem (viz obr.)



obr. 1

Vybarvení uvnitř obrysu spočívá v rozsvícení všech bodů uvnitř tohoto obrysu. Lidský zrak pak vnímá celek, ale počítač "vidí" obraz pouze funkcí PEEK, může tedy v každém okamžiku odpovědět na dotaz po stavu pouze jednoho bodu (rozsvícený nebo zhasnutý). Aby je mohl sledovat postupně, je třeba vytvořit odpovídající organizaci (strukturu) dat.

Při hledání nejjednodužšího algoritmu vybarvení plochy obrazce si můžeme pomoci následujícím pokusem (raději pouze v myšlenkách!): Vystřihneme si z papíru obrazec s libovolným tvarem a v libovolném bodě k němu přiložíme zapálenou zápalku. Oheň se rychle rozšíří na celou plochu papíru, nezávisle na jeho tvaru a výběru bodu zapálení, i když plamen také nemůže "vidět".

Rozdělte papírek na samostatné kousky, odpovídající bodům obrazovky. Každý zapálený kousek papíru se zachová stejně - zapálí své sousedící kousky.

Základní rozdíl mezi vybarvením plochy na obrazovce a spálením papíru spočívá v tom, že počítač se může v každém okamžiku zabývat pouze jedním bodem, avšak oheň se šíří po papíru všemi směry rovnoměrně. Jak se přesvědčíme, tento rozdíl nemusí být nezbytně nutný - úprava algoritmu "spalování" na vybarvování obrazu je v zásadě možná. V literatuře se tato metoda často nazývá metodou "požáru prérie".

Představme si, že každý rozsvícený bod zkouší rozsvítit své čtyři nejbližší sousedy - nahoře, dole, vlevo a vpravo. Jelikož každý bod je popsán obrazovými souřadnicemi, určení jeho sousedů je jednoduché - postačí k aktuálním souřadnicím přidat +1 nebo -1. Po rozsvícení každého ze sousedů bude každý z těchto bodů opět považován za výchozí, atd.

Protože v daném okamžiku se můžeme zabývat pouze jedním bodem, musíme si souřadnice ostatních zapamatovat až do chvíle jejich další potřeby. Počet bodů uvnitř obrazce může představovat tisíce i desetitisíce, je srovnatelný s objemem dostupné paměti RAM našeho mikropočítače. Při definování způsobu uchovávání souřadnic dosud neobsloužených bodů musíme více dbát na racionální hospodaření s pamětí. Každý nepotřebný bod musí být z paměti co možná nejdříve odstraněn, aby uvolnil místo pro další. Zároveň je problémem nalezení kritéria zakončení práce a rozeznání situace, kdy celá vnitřní plocha obrazce již byla vybarvena.

K přechovávání souřadnic využíváme systém úschovy dat, známý jako zásobník LIFO (Last In, First Out). Tento systém se chová podobně, jako např. hromádka knih, uložených jedna na druhé. K posledně položené knize máme bezprostřední přístup, protože leží na vrcholku hromady (zásobníku). Abychom se dostali ke knize, která leží úplně vespod (na dně zásobníku), musíme nejprve odebrat všechny, které leží nad ní.

Činnost našeho programu bude spočívat v kontrole posledně rozsvíceného bodu a zároveň všech jeho sousedů: nahoře, dole, vlevo a vpravo. Pokud některý z těchto bodů je již rozsvícen, dále se s ním nezabýváme. V opačném případě tento bod rozsvítíme

a jeho souřadnice uložíme "na hromadu"

Po ukončení kontroly sousedů jednoho bodu přejdeme ke hledání dalšího rozsvíceného bodu, jeho výběr je naprosto libovolný. Nejjednodužší bude vzít bod, nacházející se na vrcholku naší "hromady" souřadnic, zde přeci skladujeme souřadnice již rozsvícených bodů, a postupujeme stejným způsobem dále.

V jazyku BASIC nemáme ani zásobník, tím méně příkaz, který by realizoval funkci "polož na vrcholek" nebo "odeber z vrcholku zásobníku". Strukturu dat se stejnými vlastnostmi, jako má zásobník, získáme použitím tabulky Z s jednoduchou proměnnou, která přestavuje ukazatel zásobníku UZ. Tato proměnná ukazuje vždy na tu pozici tabulky, na které byla vepsána nová hodnota. Každé odvolání do tabulky Z se bude řídit podle pozice, dané hodnotou UZ. Při ukládání na zásobník se hodnota UZ zvětšuje o jednu před zápisem nového údaje do tabulky, během odebírání souřadnic ze zásobníku (tabulky) se proměnná UZ zmenšuje o jednu po provedení příkazu.

Dále uvedený program znázorňuje princip vybarvování obrysů:

PROGRAM 43

```
 10 PLOT 118,90: DRAW 20,0,5.35
 20 DRAW 0,-30: DRAW -20,0: DRAW 0,30
 30 LET x=128: LET y=50: GO SUB 100
 40 STOP
100 DIM z(300,2): LET uz=0
110 GO SUB 170: IF uz=0 THEN RETURN
120 LET x=z(uz,1): LET y=z(uz,2): LET uz=uz-1
130 LET y=y+1: GO SUB 170
140 LET y=y-2: GO SUB 170
150 LET y=y+1: LET x=x-1: GO SUB 170
160 LET x=x+2: GO TO 110
170 IF POINT (x,y)=1 THEN RETURN
180 PLOT x,y: LET uz=uz+1
190 LET z(uz,1)=x: LET z(uz,2)=y
200 RETURN
```

Příkazy na řádcích 10 až 40 demonstrují práci vlastního podprogramu vybarvení, který začíná od řádku 100. Před vyvoláním podprogramu je zapotřebí proměnným X a Y přiřadit souřadnice libovolného bodu, ležícího uvnitř obrysu.

Na začátku podprogramu je deklarována tabulka (pole) Z, která představuje zásobník, a ukazatel zásobníku UZ je vynulován (zásobník je zatím prázdný). Jak je z deklarace vidět, počet vrstev zásobníku nemůže být větší než 300.

Od řádku 170 vyzkouší podprogram bod se souřadnicemi X,Y, zda je rozsvícený či zhasnutý. Pokud načte rozsvícený bod, následuje bezprostřední návrat, v opačném případě je bod rozsvícen a jeho souřadnice jsou uloženy do zásobníku (viz řádky 180 a 190).

Po obsloužení sousedů jsou ze zásobníku odebírány souřadnice bodu, nacházejícího se na jeho vrcholku (řádek 120), poté následují zkoušky rozsvícení všech jeho čtyřech sousedů (řádky 130 až 160).

Nyní si představme, že bychom změnili podmínku na řádku 170 a porovnávali hodnotu funkce POINT s nulou, místo s jedničkou, a zároveň zaměnili příkaz PLOT za PLOT INVERSE 1, v tom případě bychom obdrželi program, který vymaže libovolný zaplněný obrazec z obrazovky. Tato varianta činnosti lépe připomíná spalování papírového lístku. Pochopitelně, že proměnné X a Y musí v okamžiku vyvolání podprogramu ukazovat na libovolný zaplněný (rozsvícený) bod uvnitř obrazce.

Jednoduchost našeho programu je zaplacena velkou spotřebou paměti. Výška zásobníku (t.j. počet v něm uložených prvků) roste velmi rychle. Již při rozměrově nevelkých obrazcích může snadno dojít k přeplnění tabulky Z. Zvětšování rozměru tabulky Z v deklaraci je nevhodné řešení, protože organizace použité paměti je dostupná jak pro program, tak i pro data.

Výhodnější je uspořádat zásobník pomocí abecední tabulky, místo číselné. Vycházíme z toho, že žádná ze souřadnic obrazu nemůže v žádném případě přestoupit hodnotu 255, takže takto je možno umístit více souřadnic do jednotlivých bajtů. Jediná nevýhoda je nutnost zápisu dat do tabulky pomocí funkce CHR$ a jejich čtení funkcí CODE.

Takové řešení je znázorněno v následujícím podprogramu PAINT, tímto způsobem je umožněno zvětšit přípustnou výšku zásobníku při nezměněných požadavcích na rozsahu paměti, kterou tabulka zabírá.

PROGRAM 44

```
  10 CIRCLE 128,70,20: CIRCLE 126,72,14
  20 LET x=112: LET y=71: LET dp=0
  30 GO SUB 9770: PRINT "ZAPLNENO: ",ip
  40 LET x=112: LET y=71: LET dp=1
  50 GO SUB 9770: PRINT "VYMAZANO: ";ip
  60 STOP
9770 REM PAINT (x,y,dp)
9771 DIM z$(1500,2): LET uz=0: LET ip=0
9772 GO SUB 9780: IF uz=0 THEN RETURN
9773 LET x=CODE z$ (uz,1)
9774 LET y=CODE z$ (uz,2)
9775 LET uz=uz-1
9776 LET y=y+1: GO SUB 9780
9777 LET y=y-2: GO SUB 9780
```

```
9778 LET y=y+1: LET x=x-1: GO SUB 9780
9779 LET x=x+2: GO TO 9772
9780 IF x<0 OR x>255 THEN RETURN
9781 IF y<0 OR y>175 THEN RETURN
9782 IF POINT (x,y)=dp=0 THEN RETURN
9783 PLOT INVERSE dp;x,y
9784 LET uz=uz+1: LET ip=ip+1
9785 LET z$ (uz,1)= CHR$ x
9786 LET z$ (uz,2)= CHR$ y
9787 RETURN
```

Krátký demonstrační program na řádcích 10 až 60 předvádí možnosti podprogramu PAINT. Zde těchto možností máme více, než v předešlém příkladě, jelikož tento podprogram obsahuje navíc několik vylepšení. Především hranice obrazu jsou interpretovány jako prvek obrysu - to nám umožnují příkazy na řádcích 9780 a 9781.

Podprogram je možno přepnout do dvou funkcí - vybarvování a mazání. Výběr činnosti se řídí pomocí proměnné, označující druh práce - dp. Pokud v okamžiku vyvolání proměnná dp obsahuje nulu, bude obrys vybarven. Je-li jeho obsahem jednička (dp=1), program bude z obrazu dříve zaplněné body vymazávat. V obou případech musí proměnné X a Y v okamžiku vyvolání ukazovat na některý bod, který patří do prostoru obrazce určeného k vybarvení nebo k vymazání.

Mimo čistě grafických funkcí je také možné používat podprogram PAINT jako obrazový planimetr, díky jeho proměnné ip. Tato proměnná je v okamžiku vyvolání vynulována a pak postupně zvětšuje svůj obsah o jednu při každém rozsvícení nebo zhasnutí bodu (viz řádek 9784). Po návratu z podprogramu obsahuje proměnná ip počet bodů, ať již rozsvícených, nebo zhasnutých - ve druhém případě jsou ale započítány i body obrysu. Není tedy proč se divit, že po vybarvení a následném smazání nakresleného obrazce obdržíme různé hodnoty proměnné ip (program 44, řádky 30 a 50). Druhý údaj je vždy vyšší než první právě o počet bodů, tvořících obrys obrazce (např. při prvém vyvolání ohlásí:

ZAPLNENO 530, při druhém vyvolání ohlásí VYMAZANO 772.

Mimo přípravnou činnost je podprogram PAINT v jazyce BASIC značně pomalý. Zvýšit rychlost provádění tělo funkce je možné tehdy, opustíme-li BASIC a nahradíme jej programem ve strojovém kodu. Dále uvedený podprogram realizuje tytéž funkce, jako podprogram PAINT v jazyce BASIC, ale pracuje nesrovnatelně rychleji (jako ostatně všechny strojové programy).

PROGRAM 45

```
9720 REM PAINT SUPER
9721 DATA 50000,9721
9722 DATA "AF1803E011600180339"
9723 DATA "16FFAF2A8D56228F5CE4"
9724 DATA "D95F010000D9ED4B65AF"
9725 DATA "5C21C0FFED42E5DDE10E"
9726 DATA "06FFC5ED4BB05CCDAA85"
9727 DATA "C3C1042823DDE5DD39AB"
9728 DATA "DDE1D2151F78FEB0DCC6"
9729 DATA "AAC33EFE8047DCAAC3B9"
9730 DATA "040CC4AAC33EFE814F4D"
9731 DATA "38D818D9D9C5D9C1C902"
9732 DATA "C5CDAA223C47A44F1EF2"
9733 DATA "01CB0B10FC7EAAA37927"
9734 DATA "C1C0D903A3D920057E7C"
9735 DATA "AAB3AA77D5CDDB0BD1D7"
9736 DATA "E1C5E98F",""
```

Podprogram je uložen do paměti od adresy 50000 a bez úprav není relokovatelný, jeho délka je 129 bajtů.

Aby jste měli možnost jej umístit v libovolném místě paměti, je nutno použít doplněk k zaváděcímu programu, který přeadresuje příslušné bajty. Provedeme RESTORE 9720 a do řádku 9721 vepíšeme novou adresu počátku ukládání programu. Zaváděcí program ještě doplníme

PROGRAM 46

```
200 GO SUB 9990: LET a=a-129
210 LET w2=INT ((a+90)/256)
220 LET w1=a+90-256*w2
230 POKE a+44,w1: POKE a+45,w2
240 POKE a+63,w1: POKE a+64,w2
250 POKE a+70,w1: POKE a+71,w2
260 POKE a+75,w1: POKE a+76,w2
```

Program ve strojovém kodu bude nyní vložen do paměti počínaje adresou, zadanou pomocí standardního zaváděcího programu (loaderu) od řádku 9990 - viz program 13.

Po zavedení programu nyní následují příkazy POKE, které strojový kod upraví tak, aby mohl pracovat v jiné oblasti paměti, než bylo určeno původně. Je ale samozřejmostí, že před spuštěním zaváděcího programu vložíme do počítače příkaz CLEAR s argumentem alespoň o 1 nižším, než je určená adresa počátku uložení strojového kodu (v řádku 9721).

Uvedený podprogram ve strojovém kodu využívá při své činnosti t.zv. systémový zásobník (STACK). V případě jeho přeplnění se nemůže nic stát - práce programu se přeruší a na obrazovce se objeví hlášení "Out of memory" (došla paměť).

Podprogram je možno vyvolat třemi způsoby. Vyvolání od počáteční adresy (adrp) způsobí souvislé vybarvení obrazce. Po vyvolání od adresy adrp+3 způsobí také vybarvení, tentokrát ale nikoli souvislé, nýbrž síťovým vzorkem. Vyvolání podprogramu od adresy adrp+9 způsobí vymazání vybarveného obrazce.

Pokud požadujeme síťování, máme k dispozici dva různé vzory sítě. Vzor můžeme přepnout vložením hodnoty 172 do adresového místa adrp+96, zpět k původnímu vzoru se dostaneme vložením hodnoty 164 (pomocí POKE v BASIC je tyto hodnoty možno kdykoli změnit).

Startovní bod podprogramu se určuje vepsáním souřadnice X na adresu 23728 a souřadnice Y na adresu 23729. Obě tyto adresy leží ve skutečnosti v rozsahu systémových proměnných, ale normálně nejsou využívány (byly určeny pro uschování vektoru

nemaskovatelného přerušení, NMI, v manuálu jsou označeny jako "nepoužito").

Dále uvedený příklad demonstruje provádění tohoto podprogramu a předvádí i jeho další možnosti použití.

PROGRAM 47

```
10 CLEAR 49999: GO SUB 9990
20 FOR r=1 TO 3
30 CIRCLE 128,60+7*r,r*19
40 NEXT r
50 POKE 23728,128: POKE 23729,130
60 RANDOMIZE USR 50000
70 POKE 23729,110: RANDOMIZE USR 50003
80 POKE 50096,172
90 POKE 23729,70: RANDOMIZE USR 50003
100 CIRCLE 33,87,30: POKE 23728,40
110 PRINT "ZAPLNENO: ";USR 50000;" BODU."
120 PAUSE 50
130 PRINT "VYMAZANO: ";USR 50009;" BODU."
140 STOP
```

Strojový program probíhá velmi rychle a nezabírá paměť programu a proměnných v oblasti BASIC.

KRUHOVÉ DIAGRAMY.

Vyjádření složitějších údajů nebývá příliš srozumitelné, pokud není znázorněno v sugestivním, přehledném tvaru. Je to vždy tehdy, jestliže je zapotřebí docílit rychlého přehledu a srovnání několika různých údajů. Jako nejvýhodnější se jeví vyjádření pomocí odpovídajícího grafického znázornění, zvláště dobře čitelné bývají kruhové diagramy.

Jejich princip spočívá v tom, že každé z porovnávaných hodnot se přiřadí kruhová výseč, jejíž úhel je přímo úměrný

velikosti zobrazovaného údaje. Diagram pak připomíná dort rozdělený na sektory s různou velikostí.

Představme si, že disponujeme údaji představujícími počet kalorií obsažených v devíti druzích potravin. Tyto údaje vložíme do souboru DATA a zamýšlíme podle nich vytvořit kruhový diagram, za účelem lepší evidence, které prvky našeho jídelníčku mají největší vli na prodlužování našeho pásku u kalhot.

Zvolíme si proměnnou PP, která bude představovat počet prvků diagramu (zde tedy 9). Všechny údaje pak převedeme ze souboru DATA do pole tabulky X pomocí smyčky FOR....NEXT.

PROGRAM 48

```
  10 DIM x(30)
  20 LET pp=9
  30 FOR i=1 TO pp
  40 READ x(i)
  50 NEXT i
1000 REM KRUHOVY DIAGRAM
1010 CIRCLE 92,84,83
1020 LET alfa=0: LET s=0
1030 FOR i=1 TO pp
1040 LET s=s+x(i)
1050 NEXT i
1060 FOR i=1 TO pp
1070 LET alfa=2*PI/s*x(i)+alfa
1080 PLOT 92,84
1090 DRAW 83*COS alfa,83*SIN alfa
1100 NEXT i
2000 DATA 11,30,50,20,30,90,25,44,9
```

Příprava diagramu začíná na řádku 1010 nakreslením kružnice, představující obrys "dortu". Dále je proveden součet všech prvků a výsledek je uložen do proměnné S. Nyní již můžeme

začít "dort" dělit. Úhel, jaký budou svírat hranice oddělující sousední sektory, je vypočten na řádku 1070. Tento úhel svou velikostí odpovídá hodnotě daného prvku, která je uložená v proměnné S. Po vypočtení odpovídajícího úhlu postačí z bodu, označujícího střed kružnice, vykreslit úsečku k jejímu okraji.

Holový diagram je zapotřebí také popsat. Nejjednodužší se jeví každý sektor postupně označit písmenem abecedy, počínaje od "A". Po pravé straně diagramu sestrojíme tabulku, ve které budou podíly každého ze sektorů uvedeny číselně v procentech. Doplníme tedy ještě následující řádky:

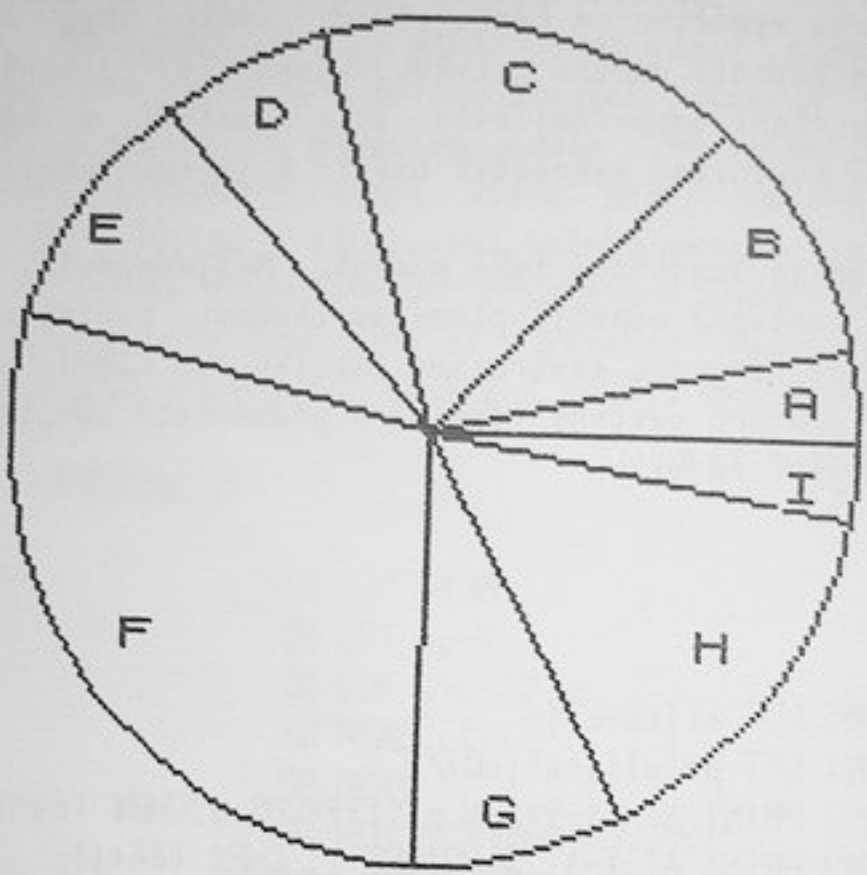
PROGRAM 49

```
1065 LET alfa0=alfa
1091 LET p=(alfa+alfa0)/2
1092 PRINT AT 11-9*SIN p,11+9*COS p;CHR$ (64+i);
1093 PRINT AT i-1,21;BRIGHT 1; CHR$ (64+i);
1094 PRINT " = ";INT (1000*x(i)/s)/10;" %";
```

V tomto doplňku programu jsme si zavedli pomocnou proměnnou alfa0, která nám uschovává úhel, pod kterým byla vykreslena úsečka, ohraničující předešlý sektor. Tuto proměnnou použijeme v řádku 1091 k vyznačení polohy osy symetrie každého sektoru. Tím je určena základní poloha místa, na které vytiskneme písmeno, odpovídající příslušnému sektoru (viz řádek 1092).

Protože tisk hodnot má pouze orientační charakter, nebylo by v tomto případě účelné vypisovat procentní rozdíly na příliš velký počet desetinných míst, úplně nám postačí jen jedno desetinné místo. Tuto úpravu program provádí na řádku 1094.

Příklad holového kruhového diagramu ukazuje obrázek.



A=3.5 %
B=9.7 %
C=16.1 %
D=6.4 %
E=9.7 %
F=29.1 %
G=8 %
H=14.2 %
I=2.9 %

Kruhové diagramy jsou velmi názorné, pokud ovšem počet zobrazovaných prvků nepřekračuje 10 - 13 a hodnota žádného z nich není menší než asi 2%.

VYKRESLENÍ PRUBĚHU FUNKCE JEDNÉ PROMĚNNÉ.

Možnost automatického vykreslení průběhu zadaných funkcí je velmi potřebná, pokud požadujeme rychlý a názorný přehled o této funkci. Vykreslený průběh je mnohem názornější pro pochopení, než samotná suchá rovnice. V praxi nejčastěji používáme funkci jedné proměnné typu $y=f(x)$, znázorněnou v pravoúhlých souřadnicích.

Zásada počítačového kreslení funkcí je stejná, jako u ručního kreslení. Spočívá v zakreslení změn nezávisle proměnné (argumentu funkce) rozdělené na určitý počet úseků, a dále vypočítat hodnotu funkce v každém z mezních bodů, výsledný bod pak přenést na papír nebo na obrazovku. Měřítko stupnice osy X je nejlépe svěřit počítači, aby rozdíl změn argumentu byl na obrazovce co nejlépe rozlišitelný.

Pokud se jedná o volbu měřítka osy Y, zde jsou použitelné dvě metody. Při prvé z nich počítač zkoumá funkci v celém rozsahu změn nezávisle proměnné "nanečisto" ještě před vlastním kreslením, přičemž nalezne minimální a maximální hodnoty, na jejichž základě je vybráno měřítko stupnice a posunutí na ose Y tak, aby nákres vyplnil celou obrazovku. Tato metoda je sice výhodná, ale na druhé straně zase neumožnuje důkladněji prohlédnout průběh zajímavých úseků křivky, ani výběr daného okénka, aby byly na obrazovce vidět např. osy souřadnic.

Druhá metoda spočívá v tom, že uživatel sám zadá přípustnou velikost změn proměnné Y, tím definuje souřadnici Y mezi horním a dolním okrajem obrazovky. Větší volnost ve výběru základu křivky je zde vykoupena omezením vstupního vkládání možných hodnot funkce. Největším problémem je možnost výskytu takových hodnot funkce, jejichž umístění v definovaném okénku na obrazovce již nebude možné a pak dojde k chybě v průběhu provádění příkazu PLOT nebo DRAW s následným přerušením programu doprovázeným chybovým hlášením. Ještě horší situace nastane během vykreslování průběhu funkcí, které mají během změn argumentu nespojité body (např. funkce y = 1 / x pro X z intervalu (-1,1)). Programy s automatickou tvorbou osy Y zde nepracují vůbec.

Jako další je uveden jednoduchý program, který umožnuje získat křivku průběhu prakticky libovolné funkce typu y=f(x) a zároveň umožnuje uniknout problémům s nespojitými body a úseky průběhů, vybíhajícími mimo obrazovku.

PROGRAM 50

```
10 REM PRUBEH FUNKCE y=f(x)
20 INPUT "Nejmensi hodnota x: ";xmin
30 INPUT "Nejvetsi hodnota x: ";xmax
40 INPUT "Nejmensi hodnota y: ";ymin
50 INPUT "Nejvetsi hodnota y: ";ymax
60 LET deltax=(xmax - xmin)/256
70 LET skalax=255/(xmax - xmin)
80 LET skalay=175/(ymax - ymin)
90 CLS
100 LET x=skalax * xmin
110 LET y=skalay * ymin
120 IF x>=0 AND x<=255 THEN PLOT x,0: DRAW 0,175
130 IF Y>=0 AND y<=175 THEN PLOT y,0: DRAW 255,0
140 LET w1=1: LET w2=0
150 FOR x=xmin TO xmax STEP deltax
160 LET x2= INT (skalax * (x-xmin)+ .5)
170 LET w2= FN B (x)
180 IF w2=0 THEN LET y= FN Y (x)
190 IF y<ymin OR y>ymax THEN LET w2=1
200 IF w2<>0 THEN LET w1=1: GO TO 260
210 IF w1=1 THEN GO TO 250
220 LET y2= INT (skalay * (y-ymin)+ .5)
230 PLOT x1,y1: DRAW x2-x1,y2-y1
240 LET y1=y2: GO TO 260
250 LET y1= INT (skalay * (y-ymin)+ .5): LET w=0
260 LET x1=x2
270 NEXT x
998 DEF FN Y (x) =(2-x)/((x-1)*(x-3))
999 DEF FN B (x) =(x=1) OR (x=3)
```

Na počátku si program vyžádá přípustný rozdíl proměnných X a Y. Dále je vypočten základní přírustek proměnné X (DeltaX) tak, aby rozdíl změn argumentu byl dělitelný 255. Dílky stupnic

pro obě osy (SkalaX, SkalaY) jsou vybrány tak, aby byl využit celý povrch uživatelského okénka.

Po smazání obrazovky (CLS na řádku 90) program vypočítá obě základní osy. Pokud se ukáže, že daná osa se do okénka vejde, je vykreslena (řádky 120 a 130). Na řádku 150 začíná vlastní smyčka počítající postupně hodnoty funkce a kreslící úseky křivek. Základ kreslení funkce je definován na řádku 998 a samozřejmě může být změněn podle požadavků uživatele, případně je možno číst funkci jako řetězcovou proměnnou z klávesnice příkazem INPUT s následujícím vyhodnocením její hodnoty funkcí VAL.

Funkce FN B na řádku 999 plní svůj účel v případech funkce s nespojitými body. Základ této funkce je třeba vybrat tak, aby ve všech bodech nespojitosti FN B byl různý od nuly (FN Y v programu 50 má tyto body dva: pro x=1 a x=3). Tam, kde je funkce spojitá v celém rozsahu změn argumentu, stačí vložit:

DEF FN B (x) =0

Hodnota funkce FN B je zjišťována ještě před vyčíslením funkce FN Y a nedopustí vyběhnutí funkce v nespojitých bodech, což by vedlo k chybám. Ukazatelem je proměnná W2.

Po zjištění hodnoty funkce následuje kontrola, zda se odpovídající bod vejde do definovaného okénka (řádek 190). Pokud ne, bod není vykreslen a proměnné W2 je přidělena nenulová hodnota. Její hodnota je nulová pouze tehdy, když průběh funkce bude vypočten v pořadku a bod se vejde do okénka.

Proměnná W1 dovolí rozeznávat situace, ve kterých není možno spojit bod, umístěný v okénku, s předešlým bodem. Pokud proměnná W1=1, znamená to, že předešlý bod ležel mimo okénko a tedy jej není možné spojit s dalším pomocí příkazu DRAW. Pouze v případech, kdy W1 i W2 jsou současně rovny nule, (t.j. předešlý i akluální bod se vejdou do okénka), může dojít k propojení těchto bodů souvislou čarou (řádky 220 až 240).

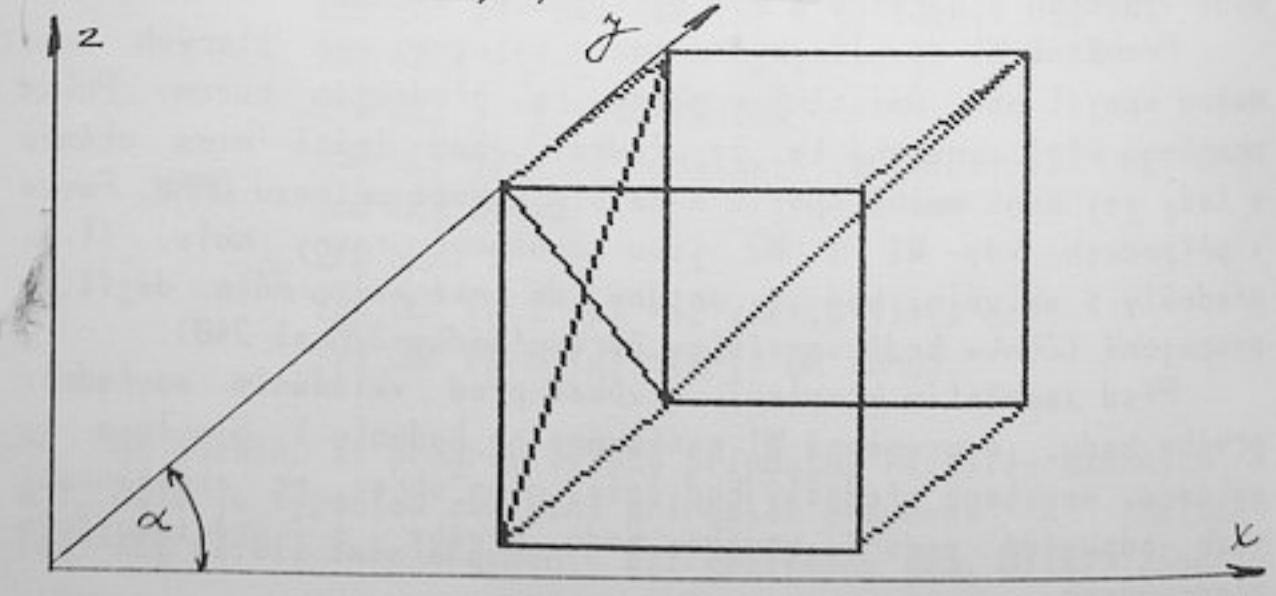
Před započetím kreslení, a vůbec před vkládáním souřadnic prvého bodu, je proměnná W1 nastavena na hodnotu 1. Simuluje to situaci, ve které předešlý bod ležel mimo obraz, ve skutečnosti však odpovídá zadání prvého bodu kresby s neexistujícím předchůdcem.

ZÁKLADY PSEUDOPROSTOROVÉ GRAFIKY

Třírozměrná grafika na obrazovce počítače je velice efektní a často vzbuzuje údiv. Z praktického hlediska je pseudoprostorové znázornění různých objektů: budovy, části strojů, auta, grafy funkcí více proměnných atd. velmi užitečné a dovoluje lepší představu o daném objektu, než jeho technický výkres či vzorec funkce.

Abychom si mohli představit trojrozměrný objekt na rovině obrazovky, musí být jeho projekce dokonalá. V zásadě rozlišujeme dva druhy prostorové projekce: rovnoběžnou (axonometrickou) a perspektivní (ústředního pohledu). Tato také nakonec nejlépe odpovídá přirozenému způsobu vidění a dává věrnější zobrazení, na druhé straně ale vyžaduje značně složité výpočty a výběr počátečních podmínek, např. polohu imaginárního pozorovatele. Zde se tedy budeme zabývat méně komplikovanou rovnoběžnou projekcí.

Axonometrická projekce představuje objekty tak, že rovnoběžné čáry na modelu zůstanou rovnoběžné i na zobrazení. Proporce rovnoběžných úseček, vyznačené v prostoru osami X a Z jsou zachovány, délky ostatních úseček budou zkráceny v závislosti na úhlu odchylky každé z nich - viz obrázek.



obr. 2

V třírozměrném zadání souřadnic je každý bod určen souřadnicemi X, Y, Z. Při prostorovém pohledu musí být každá z těchto trojic přenesena do dvojice obrazových souřadnic Xo a Yo. Vzhledem k tomu, že úsečky rovnoběžné s osou Y podléhají zkrácení o předem zadaný součinitel zkrácení K a úhel alfa je znám, zjistíme, že celý převod je jednoznačně určen pomocí součinitele K a úhlu alfa, a že je možno použít následujících vzorců pro určení obrazových souřadnic.

$$Xo = X + Y * K * \cos(alfa) + Xo0$$
$$Yo = Z + Y * K * \sin(alfa) + Yo0$$

kde Xo0 a Yo0 jsou konstanty, které určují polohu tělesa v okénku vzhledem k průsečíku souřadnic X Y Z.

Uvedené rovnice jsou výchozím bodem k sestavení jednoduchého programu, který na obrazovce znázorní axonometrický pohled na libovolnou třírozměrnou konstrukci.

PROGRAM 51

```
  1 REM TROJROZMERNA GRAFIKA
 10 DIM x(100): DIM y(100): DIM z(100)
 20 LET k= .6: LET alfa= .7
 30 LET sk=7
 40 LET kx=k*COS alfa
 50 LET ky=k*SIN alfa
 60 LET n=0
 70 READ w: IF w>=1E37 THEN GO TO 100
 80 LET n=n+1: LET x(n)=w: READ y(n), z(n)
 90 GO TO 70
100 CLS
110 READ a: IF a=0 THEN GO TO 200
120 READ b: LET xo=FN X (a): LET yo=FN Y (a)
130 PLOT xo,yo: DRAW FN X (b)-xo, FN Y (b)-yo
140 GO TO 110
4800 DEF FN X (a)=sk*(x(a)+kx*y(a))+128
4900 DEF FN Y (a)=sk*(z(a)+ky*y(a))+87
```

```
5000 DATA -5,-5,-5,-5,5,-5,5,5,-5,5,-5,-5
5010 DATA -5,-5,5,-5,5,5,5,5,5,5,-5,5
5020 DATA 1E37
6000 DATA 1,2,2,3,3,4,4,1,5,6,6,7,7,8,8,5
6010 DATA 1,5,2,6,3,7,4,8
6020 DATA 1,6,2,7,2,5
6030 DATA 0
```

Převod souřadnic X Y Z na obrazové souřadnice je zde realizován pomocí definovaných funkcí FN Y a FN X. Argumentem funkce je číslo vrcholu, jehož souřadnice právě kreslíme. Proměnná SK je součinitel měřítka, dovolující zvětšování i zmenšování vykreslovaného obrazce dle potřeby. Současně je možno proměnnou SK, součinitel zkrácení K a úhel alfa upravovat, podle podmínek pro daný pohled nejvhodnějších.

Informace o vlastním tělese a o pohledu na něj je uložena do dvou bloků DATA. První blok, který začíná na řádku 5000, obsahuje souřadnice vrcholů, které jsou jsou postupně očíslovány počínaje 1. Souřadnice vrcholků musí být v bloku DATA zapsány ve správném pořadí - vrcholek číslo 1 má souřadnice (-5,-5,-5), vrcholek číslo 4 (5,-5,-5), atd. Blok dat určujících vrcholy je zakončen číslem nejméně 1E37 (řádek 5020). Tento údaj plní úlohu "hlídače", jeho načtení označuje, že již nejsou žádné další vrcholy - čtení dat je ukončeno.

Souřadnice vrcholů jsou na počátku práce programu čteny z bloku dat a uloženy do třech tabulek X, Y a Z. Index každé souřadnice v tabulce odpovídá číslu vrcholu, určeného těmito souřadnicemi. Během zapisování souřadnic do tabulek je odpočítáván počet vrcholů (proměnná N).

Vykreslování pohledu na těleso začíná na řádku 100 vymazáním obrazovky, následuje kreslení vrcholů a jejich spojnic. K tomu ale potřebujeme další dodatečné informace o tom, které vrcholy mají být navzájem propojeny čarou. Tyto informace obsahuje druhý blok DATA začínající od řádky 6000. Zde jsou uvedeny dvojice číslic, které označují vrcholy, jež mají být

spolu propojeny, každá dvojice definuje jednu spojnici. Například prvá dvojice 1,2 určuje, že je zapotřebí vykreslit čáru od vrcholu číslo 1 k vrcholu číslo 2, atd.

Data, uvedená v našem příkladu popisují krychli, ve které jsou dodatečně do jedné ze stěn dokresleny dvě úhlopříčky a do druhé stěny jedna úhlopříčka. Blok dat, určujících propojky, končí také "hlídačem". V tomto případě je to Ø - vrcholek s tímto číslem se nemůže vyskytovat.

Po načtení obou čísel vrcholů jsou vypočteny obrazové souřadnice prvého (řádek 12Ø) a následuje vykreslení úsečky od prvého k druhému vrcholu (řádek 13Ø). Převod souřadnic provádí definované funkce FN X a FN Y. Konstanty 128 a 87 jsou bodem průsečíku obou souřadnic, je možno je i změnit podle požadavků, pokud je toho zapotřebí.

V případě, že chceme vykreslit tvar jiného tělesa, postačí změnit údaje v obou blocích DATA. Výhodné je nakreslit dané těleso nejprve na list papíru, očíslovat vrcholy a teprve pak vkládat údaje do bloků DATA. Také je možné pokusit se o sestavení programu, umožnujícího kreslit přímo na obrazovce, je to jen o něco málo složitější. Po vložení nových bloků dat nesmíme zapomenout na ukončení každého z nich ukazatelem konce - t.zv. "hlídačem".

Náš program správně vykreslí obrysy tělesa se známými souřadnicemi vrcholů, ale tím jsou jeho možnosti dost omezené. Nanejvýše je možné experimentovat se změnou měřítka a souřadnicemi projekce.

My bychom ale požadovali také otáčení vykresleného tělesa kolem každé ze základních os souřadnic, abychom si jej mohli prohlédnout ze všech stran. Takže nyní budeme potřebovat příslušný převod proměnných, které označují jednotlivé souřadnice.

Připusťme, že požadujeme otočit těleso kolem osy X o úhel betaX. Souřadnice X těchto vrcholů se tedy nemění, nové souřadnice Y' a Z' jsou určeny vztahy:

```
X'=X
Y'=Y * COS BetaX + Z * SIN BetaX
Z'=-Y * SIN BetaX + Z * COS BetaX
```

Tyto vzorce je možno nalést v učebnicích geometrie, nebo se pokusit o jejich vlastní odvození. Postačí provést jednoduchý situační náčrt.

V případě otočky tělesa kolem osy Y budou převodní vzorce podobné:

```
X'=X * COS BetaY + Z * SIN BetaY
Y'=Y
Z'=-X * SIN BetaY + Z * COS BetaY
```

Analogicky i pro otáčení kolem osy Z:

```
X'=X * COS BetaZ + Y * SIN BetaZ
Y'=-X * SIN BetaZ + Y * COS BetaZ
Z'=Z
```

Každou obrátku tělesa v prostoru je možno složit ze třech vzájemně nezávislých otočení kolem každé z os. Takže nám již nezbývá nic jiného, než doplnit náš program 51 o uvedenou možnost otáčení definovaného tělesa v prostoru.

Program se nás po vykreslení tělesa zeptá na úhly otočení kolem každé ze třech os souřadnic:

PROGRAM 52

```
200 INPUT "Uhel otoceni kolem osy X: ";bx
210 INPUT "Uhel otoceni kolem osy Y: ";by
220 INPUT "Uhel otoceni kolem osy Z: ";bz
230 LET sx=SIN (PI*bx/180): LET cx=COS (PI*bx/
    180)
240 LET sy=SIN (PI*by/180): LET cy=COS (PI*by/
    180)
250 LET sz=SIN (PI*bz/180): LET cz=COS (PI*bz/
    180)
```

```
260 FOR i=1 TO n
270 LET y1=y(i): LET z1=z(i)
280 LET y(i)=y1*cx+z1*sx: LET z(i)=-y1*sx+z1*cx
290 LET x1=x(i): LET z1=z(i)
300 LET x(i)=x1*cy+z1*sy: LET z(i)=x1*sy+z1*cy
310 LET x1=x(i): LET y1=y(i)
320 LET x(i)=x1*cz+y1*sz: LET y(i)=-x1*sz+y1*cz
330 NEXT i
340 RESTORE 6000: GO TO 100
```

Program umožnuje vkládat požadované úhly ve stupnové míře. Jelikož hodnoty žádaných sinů a kosinů budeme potřebovat vícekrát a výpočet trigonometrických funkcí je časově náročný, budou hodnoty těchto funkcí počítány jen jednou a uloženy jako hotové součinitele transformace (např. SX = SIN Beta X, CX = COS Beta X atd.). Během výpočtu součinitelů je úhlové měřítko současně přepočteno na měřítko v radianech.

Nyní můžeme přistoupit k přepočtu souřadnic, čili k vlastnímu otáčení tělesa. Pro jednoduchost provedeme pouze tři otočení: kolem osy X (řádky 270 a 280), osy Y (řádky 290 a 300) a konečně kolem osy Z (řádky 310 a 320). Přepočet probíhá podle dříve uvedených vzorců. Operace se provádí samostatně pro každý vrcholek, po jejím zakončení je možno se vrátit do podprogramu kreslení tělesa na obrazovce. Za tímto účelem je ale nutno přesunout ukazatel čtení souboru DATA na počátek bloku, který popisuje průběh (příkaz RESTORE 6000 na řádku 340).

Existuje také možnost provedení otočky tělesa "jedním šupem", v jediné operaci, bez kouskování na postupné otočky kolem osy. Pak se ale příslušné vzorce komplikují a stávají se méně přehlednými, je nutností použít operace s maticemi.

Při přípravě dat těles, kterými hodláme na obrazovce otáčet, je vhodné vybírat souřadnice vrcholů tak, aby průsečík souřadnic připadl do středu daného tělesa. To nás ochrání před "vypadnutím" tělesa z obrazu během jeho otáčení. Lepší, ale zato značně komplikovanější řešení je napsat odpovídající program, který těleso automaticky vystředí.

Jiné, populární použití pseudoprostorové grafiky, je kreslení průběhu funkcí dvou proměnných. Základ je jednoduchý - změny obou argumentů (např. X a Y) se rozdělí na úseky stejné délky, a následně je vypočtena hodnota funkce pro všechny kombinace X a Y. Výsledné hodnoty se označí jako souřadnice Z a představují již dříve popsaný způsob.

Nejobtížnější zde bývá odstranění neviditelných čar. Méně elegantní, zato však jednoduché řešení spočívá v založení jednorozměrné tabulky s 256 prvky - každému z nich je přiřazena jedna z 256 možných souřadnic osy Xo. Obrazec je pak nutno kreslit od nejnižší hodnoty Y k nejvyšší. Po vyznačení vzájemné závislosti Z se počítají souřadnice obrazu Xo a Yo. Dále porovnává Yo s tou položkou tabulky, která odpovídá Xo. Pokud je Yo větší, zakreslíme bod a Yo vepíšeme do tabulky místo původní hodnoty, v opačném případě bod vynecháme. Princip této metody spočívá v tom, že body na vzdálenějších rovinách se zakryjí objekty na bližší rovině a plocha nákresu je souvislá.

Dále je uveden program, který pracuje uvedeným způsobem. Tabulka H je vlastně výše popsaná tabulka zákrytů.

PROGRAM 53

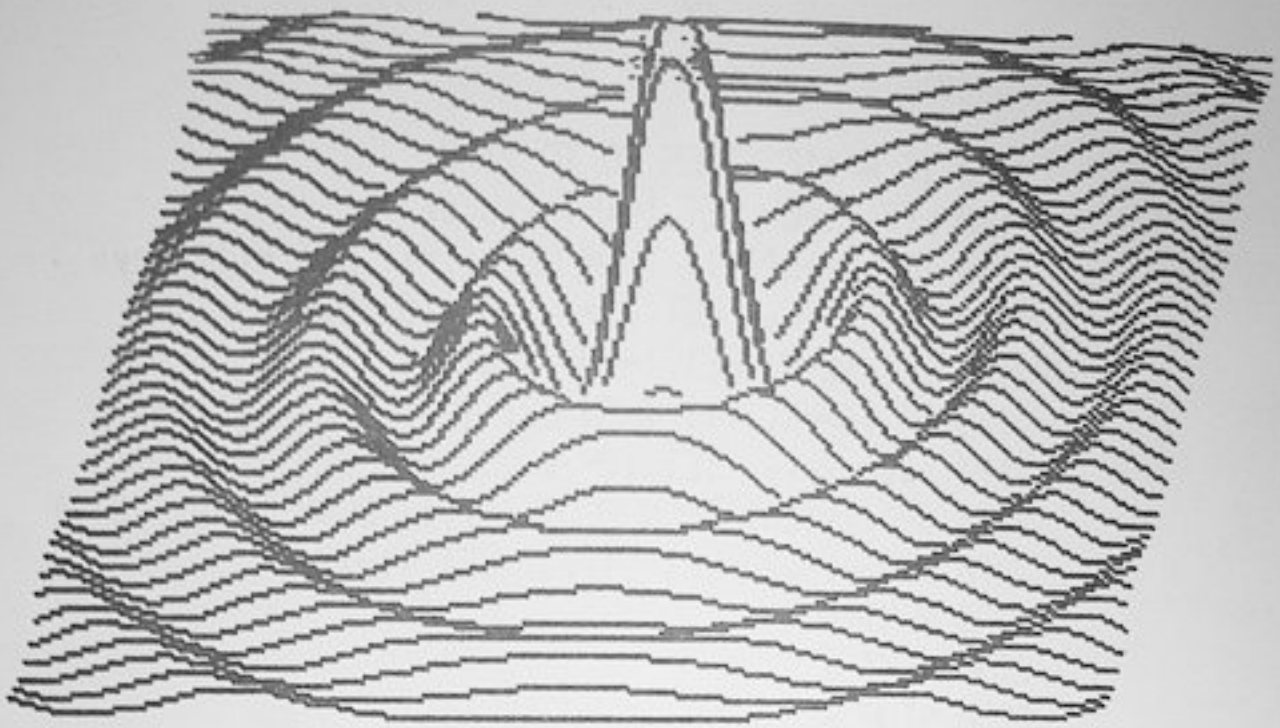
```
2000 REM Prostorovy prubeh funkce
2010 LET a=1080: LET k1=80
2020 LET w=75: LET k=0.75
2030 LET xo0=128: LET yo0=83: LET bm=PI/180
2040 LET c=k*COS (w*bm): LET s=k*SIN (w*bm)
2050 LET dx=3: LET dy=5: LET af=a/90
2060 DIM h (256)
2070 FOR I=1 TO 256: LET h(I)=-1000: NEXT I
2080 FOR g=-110 TO 110 STEP dy: LET y=g*af
2090 FOR m=-105 TO 105 STEP dx
2100 LET x=m*af: GO SUB 2400
2110 LET xo=INT (xo0+m+c*g+.5): LET yo=INT (yo0+
     s*g+z+.5)
```

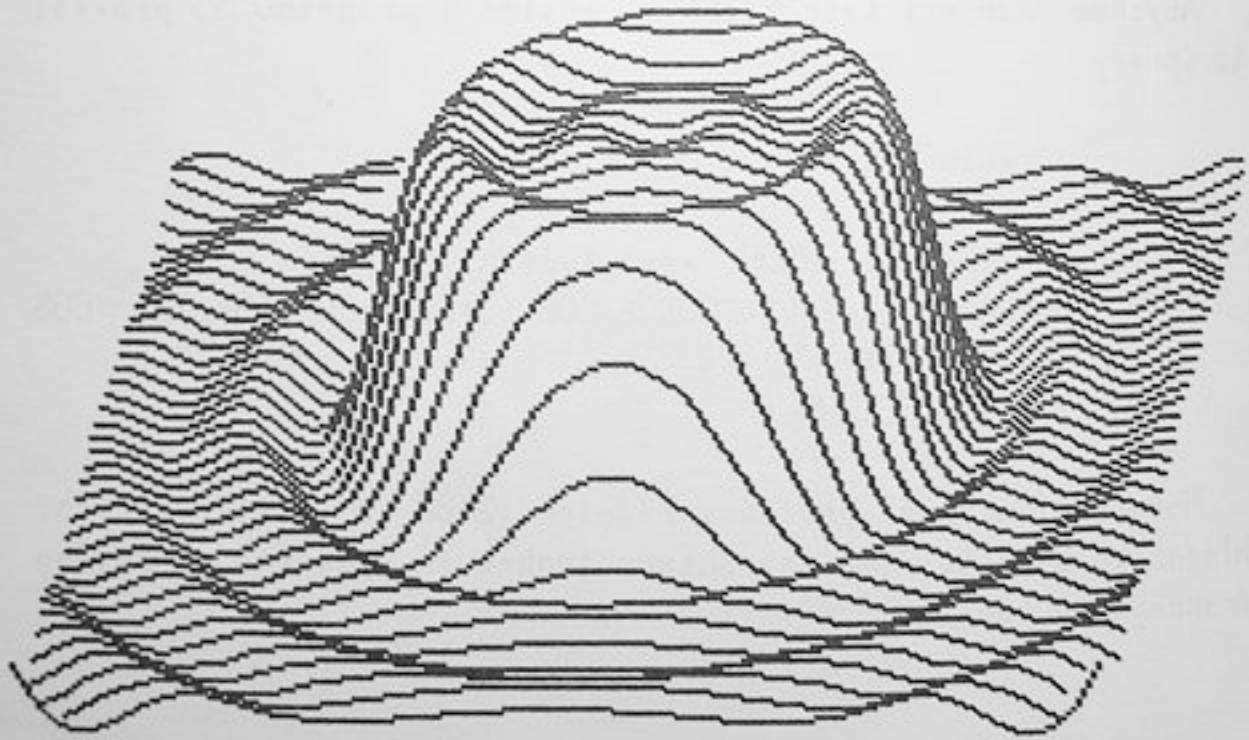
```
2130 IF m>-105 THEN GO TO 2170
2140 LET f1=0: LET l=INT (xo/dx)
2150 IF yo>=h(l+1) THEN LET f1=1: LET h(l+1)=yo
2160 LET x1=xo: LET y1=yo: GO TO 2220
2170 LET f2=0: LET l=INT (xo/dx)
2180 IF yo>=h(l+1) THEN LET f2=1: LET h(l)=yo
2190 let x2=xo. LET y2=yo
2200 IF f1*f2=1 THEN PLOT x1,y1: DRAW x2-x1,y2-y1
2210 LET x1=x2: LET y1=y2: LET f1=f2
2220 NEXT m
2230 NEXT g
2250 STOP
2400 LET r=SQR (x*x+y*y)*bm
2410 IF r=0 THEN LET z=k1: RETURN
2420 LET z=k1*SIN (r)/r RETURN
```

Abychom obdrželi tvar z obr.4, musíme v programu 53 provést tyto úpravy:

```
2010 LET a=155: LET k1=39
2020 LET w=75  LET k=0.65
2400 LET r=SQR (x*x+y*y)*bm
2410 LET z=k1*(COS r-COS (3*r)/3+COS (5*r)/5- COS
     (7*r)/7)+24
2420 RETURN
```

Hodnota funkce se počítá v řádcích 2400 a následujících. Vzhledem ke složitějšímu charakteru funkce je upuštěno od jejího podrobnějšího popisu.





PROSTOROVÉ MODELY MOLEKUL

Zajímavým a efektním použitím počítačové pseudoprostorové grafiky je znázornění skladby molekul, jednotlivé atomy molekuly jsou znázorněny jako kuličky. Způsob stínování kuliček může být různý, podle druhu jednotlivých atomů. Molekuly je možno na obrazovce znázornovat různým způsobem, otáčet jimi a prohlížet je z různých stran. Mimo znázornění molekul o známém složení je samozřejmě možné se také bavit modelováním nových molekul s dosud neznámou strukturou.

Protože nám nezáleží na znázornění molekuly v konkrétních souřadnicích, můžeme jejich projekci značně zjednodušit. Dejme tomu, že bod, ze kterého molekulu sledujeme, leží v prodloužení osy Y (obr. 3). V tomto případě se součinitel zkrácení rovná nule, což funkci převodu prostorových souřadnic na obrazové značně zjednoduší. K prostorovému otáčení molekuly využijeme již známé části programu 52.

Pro tvorbu obrazce molekuly budeme potřebovat souřadnice příslušných atomů, vyznačené v nějakém - libovolném seskupení pravoúhlých souřadnic v daném měřítku. Náš model na obrazovce musí znázornovat nejenom geometrické uspořádání atomů v molekule, ale musí také umožnit změnu jejich rozložení. Toho je možné docílit znázorněním atomů různých prvků pomocí kuliček různých druhů, které se liší stínováním ald.

Každý atom v našem modelu bude popsán čtveřicí údajů: tři souřadnice (X, Y, Z) a druh kuličky, představující daný atom. O co nám tvorba molekuly na obrazovce přináší méně starostí, o tolik jich budeme mít více s problémem maskování neviditelných částí kuliček, zacloněných ať již částečně, nebo celé dalšími kuličkami, které leží blíže k pozorovateli. Vyřešit tento problém je nezbytné pro vytvoření realistického perspektivního obrazu molekuly. Dále je zapotřebí se zaměřit na takový způsob znázornování plasticity kuliček, aby pozorovatel vnímal jejich

oblý povrch, tento dojem docílíme příslušným stínováním.

Příkladem programu v BASIC, který v dostatečné míře splnuje všechny uvedené požadavky, je program 54. Najdeme v něm více prvků, které jsme poznali již dříve.

Údaje o struktuře molekuly jsou sestaveny do čtveřic hodnot v bloku DATA, který začíná na řádku 5000. Blok opět končí "hlídačem" 1E37. Funkce, které převádějí prostorové souřadnice na obrazové, jsou značně zjednodušeny na základě našich úvah o lokalizaci pozorovatele (řádek 4800).

Tři tabulky souřadnic byly doplněny čtvrtou, pro proměnné označující typ atomů (kuliček) - R (viz řádek 10). Součinitel měřítka SK byl doplněn dalším - SR, který určuje měřítko, ve kterém jsou kresleny atomy - kuličky. Smyčka na řádcích 40 až 70 přenáší údaje z bloku DATA do tabulek a počítá zadané atomy (proměnná N).

PROGRAM 54

```
 10 REM Prostorovy model molekuly
 20 DIM x(100): DIM y(100): DIM z(100): DIM
    r(100)
 30 LET sk=25: LET sr=12
 40 LET n=0
 50 READ w: IF w>=1E37 THEN GO TO 100
 60 LET n=n+1: LET x(n)=w: READ y(n),z(n),r(n)
 70 GO TO 50
100 CLS
110 LET f=0
120 FOR i=1 TO n-1
130 IF y(i)>=y(i+1) THEN GO TO 190
140 LET t=y(i): LET y(i)=y(i+1): LET y(i+1)=t
150 LET t=x(i): LET x(i)=x(i+1): LET x(i+1)=t
160 LET t=z(i): LET z(i)=z(i+1): LET z(i+1)=t
170 LET t=r(i): LET r(i)=r(i+1): LET r(i+1)=t
```

```
180 LET f=1
190 NEXT i: IF f=1 THEN GO TO 110
200 FOR i=1 TO n
210 LET xo=FN X (i): LET yo=FN Y (i): LET  ro=sr
    *r(i)
220 GO SUB 1000
230 NEXT i
240 INPUT "Uhel otoceni kolem osy X: ";bx
250 INPUT "Uhel otoceni kolem osy Y: ";by
260 INPUT "Uhel otoceni kolem osy Z: ";bz
270 LET sx=SIN (PI*bx/180): LET cx=COS (PI*bx/
    180)
280 LET sy=SIN (PI*by/180): LET cy=COS (PI*by/
    180)
290 LET sz=SIN (PI*bz/180): LET cz=COS (PI*bz/
    180)
300 FOR i=1 TO n
310 LET y1=y(i): LET z1=z(i)
320 LET y(i)=y1*cx+z1*sx: LET z(i)=-y1*sx+z1*cx
330 LET x1=x(i): LET z1=z(i)
340 LET x(i)=x1*cy+z1*sy: LET z(i)=-x1*sy+z1*cy
350 LET x1=x(i): LET y1=y(i)
360 LET x(i)=x1*cz+y1*sz: LET y(i)=-x1*sz+y1*cz
370 NEXT i
380 GO TO 100
1000 FOR j=0 TO ro-1 STEP .5: CIRCLE INVERSE 1;xo
     ,yo,j
1010 NEXT j: CIRCLE xo,yo,ro
1020 FOR j=1 TO 150
1030 LET fi=2*PI*RND
1040 LET d=ro*(1-RND *RND *.4)-1
1050 LET a=d*SIN fi: LET b=d*COS fi
1060 PLOT xo+a,yo+b
1070 NEXT j
1080 RETURN
```

```
4800 DEF FN X (a)=sk * x(a) + 128
4900 DEF fn Y (a)=sk * z(a) + 87
5000 DATA -2.5,0,0,1,-2,0,.87,1,-1,0,.87,1
5010 DATA -.5,0,0,1,-1,0,-.87,1,-2,0,-.87,1
5020 DATA .5,0,0,1,1,.348,.348,1,2,.348,.348,1
5030 DATA 2.5,0,0,1,2,-.348,-.348,1,1,-.348,-.348
5040 DATA 1,1E37
```

Chceme znázornit několik kuliček, umístěných v různé vzdálenosti od pozorovatele. Je samozřejmé, že kuličky ležící blíže budou překrývat ty, které se nacházejí dále. Při kreslení kuliček např. na tabuli začneme od těch, které jsou umístěny nejhlouběji. Před nakreslením každé kuličky nejprve vymažeme plochu, ve které se bude nacházet. Pokud by tato plocha zabírala část objektu na tabuli již existujícího, znamená to, že tento objekt bude překryt, jakoby neviditelný, takže zůstane vymazán.

Podobně budeme postupovat na obrazovce. Začneme atomy položenými nejdále od pozorovatele, podle údajů pro osu Y. Před započetím kreslení uspořádáme tabulky X, Y, Z a R tak, aby atomy byly uspořádány za sebou podle souřadnice Y. Objekt s nejvyšší hodnotou Y bude označen číslem 1 atd.

Podprogram, třídící atomy podle hodnoty Y je na řádcích 110 až 190. Pro třídění je použit algoritmus zvaný "bublesort", bublinkové třídění. Není právě nejefektivnější, ale zato je jednoduchý. Jeho činnost spočívá v tom, že několikrát prohlédneme tabulku Y a při tom porovnáváme sousedící páry prvků: 1 s 2, 2 s 3, 3 s 4 atd., až do n-1 s n. V n-prvkové tabulce je n-1 párů sousedících prvků, viz řádek 120.

Pokud při porovnávání prvků v daném páru (řádek 130) se ukáže, že první prvek není menší než druhý, přestaneme se tímto párem zabývat a přejdeme k dalšímu. Pokud jsou tyto prvky uloženy v obráceném pořadí (menší před větším), jednoduše oba prvky spolu zaměníme. Při tom je nestačí zaměnit pouze v tabulce Y, ale musíme je stejným způsobem zaměnit i v dalších tabulkách. Jenom tak docílíme toho, že prvky všech tabulek s tímtéž indexem

patří ke stejnému bodu (atomu).

Přesun prvků v tabulkách zařídí řádky 140 až 170. Po každé operaci přesunu obdrží proměnná F hodnotu 1. Tato proměnná je před každým dalším prohledáváním tabulky vynulována (viz řádek 110). Proměnná F zachovává hodnotu 0 po ukončení prohledávání všech dvojic pouze tehdy, pokud nebyla provedena žádná změna, znamenalo by to, že se nepřekrývá ani jeden pár sousedních prvků, zkrátka - nulová hodnota F po prohlédnutí tabulky ukazuje zakončení procesu třídění. V případě, že F=1 je nutno prohlédnout tabulku znaova a případně provést další změny.

Po ukončení třídění můžeme konečně přistoupit ke kreslení kuliček - atomů ve smyčce na řádcích 200 až 230. Proměnné Xo a Yo obdrží díky funkcím FN X a FN Y hodnoty souřadnic středů atomů, proměnná Ro je rovna obrazové proměnné dané kuličky.

Vlastní kreslení provádí podprogram od řádku 1000. Začínáme stejně jako při kreslení na tabuli - mazáním uvnitř kruhového obrysu, který udává proměnná Ro. Pomáháme si při tom jakousi partyzánštinou, použitím příkazu CIRCLE INVERSE 1 a kreslíme soustředěné kruhy, a sice barvou papíru, ne barvou inkoustu.

Pochybnost ještě může vyvolat, co znamená STEP 0.5 na řádku 1000. Tento krok používáme kvůli co možná nejdokonalejšímu vyčistění plochy, jednoduchý pokus s použitím STEP 1 ukáže jasně, o co se vlastně jedná.

Kolem vyčistěné plochy vykreslíme okraj - obrys budoucí kuličky (řádek 1010). Dále nám zbývá již jen kuličku vystínovat tak, aby se vytvořil dojem, že je skutečně vypouklá. Neejednodužši způsob, jak toho docílit, je začernění okraje, třeba melodou "rozsypání krupice". Více vzdálené body se pak slévají a tvoří dojem jednolité, souvislé plochy. Odstín plochy je jednoduše přímo úměrný hustotě rozsypané krupice - vykreslených bodů.

Abychom se vyhnuli uniformitě (jednotvárnosti) kuliček, postaráme se o to, aby rozmístění černých bodů bylo skutečně náhodné. Příkladem praktické realizace této úlohy je smyčka na řádcích 1020 až 1070. Nejprve (řádek 1020) je určen úhel, pod

jakým má být bod vržen vzhledem ke středu kružnice. Dále je podobným postupem určena vzdálenost tohoto bodu od středu. Pravděpodobnost výběru příslušného úhlu z rozsahu [0 - 2PI] je vyhovující (alespoň teoreticky), ale se vzdáleností je to jinak (řádek 1040). Použitý výraz zaručuje, že proměnná nikdy nebude menší než kolem 40% proměnné okraje a tak pravděpodobnost uložení bodu v okolí obrysu je větší, než ve vnitřní ploše kružnice.

Nyní je již možné přepočítat tuto proměnnou a příslušný úhel na odpovídající obrazové souřadnice (řádek 1050) a bod přenést na obrazovku. Hustota stínování závisí na počtu opakování smyčky (řádek 1020). Místo konstanty 150 je zde možno vložit jiný vhodný součinitel, odvozený např. od Ro. Možných vylepšení je mnoho.

Program 54, podobně jako většina ostatních programů v této příručce, má za úkol pouze inspirovat programátory v jejich vlastní činnosti.

Po znázornění rozložení atomů v zadané projekci máme možnost otáčet jimi podél každé ze tří os X, Y, Z. Program se ptá na úhel otočení a upraví souřadnice stejným způsobem, jako tomu bylo v programu 52.

Údaje, uvedené v bloku DATA programu 54 popisují molekulu bisphenolu. Není vyloučena možnost změny stínování ani změny souřadnic - zde se otevírá pole k vlastním experimentům.

JEDNODUCHÁ POČÍTAČOVÁ HRA

Většin uživatelů mikropočítačů se s nimi poprvé setkala prostřednictvím her. Naprogramování dobré hry s pseudoprostorovými efekty a kvalitní animací se vymyká možnostem většiny domácích programátorů, zvlášť pokud se jedná o strojový kód.

Někdy je však možné sestavit zajímavou hru poměrně jednoduchými prostředky, ba dokonce i jen v jazyce BASIC. Pak je nutná dobrá znalost všech možností, které je daný mikropočítač schopen poskytnout.

Příkladem může být dále uvedený program, simulující na obrazovce tenisový dvorec. Závodník, hrající v pozadí je automaticky řízen počítačem, závodník u dolního okraje obrazovky (t.j. v popředí) je řízen hráčem. Řízení spočívá v takovém přemisťování závodníka doleva nebo doprava pomocí kurzorových kláves, aby se nalézal na dráze letu míčku. Servis je automatický, vždy na straně závodníka, který naposledy ztratil bod. Způsob počítání bodů je obdobný jako v tenisu -- zápas se skládá ze setů rozdělených na hry. Hru vyhrává závodník, který jako prvý překročil počet 40 bodů (čtvrtý vyhraný míček ve hře). Pro zjednodušení se zde neuvažuje hra na převahu dvou míčků. Podobně je tomu v setu, který končí v okamžiku vítězství kteréhokoli závodníka v šesté hře.

Animace, vykonávající pohyb míčku a obou závodníků, je dosaženo díky použití UDG a příkazu OVER 1.

Údaje o čtyřech tvarech, vytvářených UDG, jsou uloženy v bloku DATA (řádky 420 a 430). Okamžitě po spuštění programu jsou data přenesena za pomoci smyčky (řádek 20) do oblasti UDG, do znaků "A", "B", "C" a "D". Silueta "počítačového" závodníka je pod znakem "A", postavu závodníka řízeného hráčem máme složenou ze znaků "B" (hlava a trup) a "C" (nohy) Pod znakem "D" je v UDG nadefinován míček.

Na řádcích 50 -- 60 je vykreslena hrací plocha. Údaje o ní jsou uloženy v řádcích DATA 440 až 460, jsou to v podstatě dvojice souřadnic určující rohové body hrací plochy. Konec seznamu souřadnic je opět označen "hlídačem" - X=999. Pokud souřadnice X je menší než 999, ale větší než 299, značí to, že daným bodem začíná nový obrys (aby se nepopletl s předešlými). Skutečná hodnota X se získá odečtením konstanty 500 od údaje v souboru DATA.

PROGRAM 55

```
10 REM TENIS
20 FOR a=USR "a" TO USR "d"+7: READ x: POKE a,x
   :NEXT a
30 INPUT "Stopen obliznosti (0-3) ",st: LET st=
   (st+2)/5
40 PAPER 4: BORDER 4: INK 7: CLS
50 READ x,y: IF x<300 THEN DRAW x,y: GO TO 50
60 IF x<999 THEN PLOT x-500,y: GO TO 50
70 FOR i=72 TO 84 STEP 2: PLOT 70,i: DRAW 115,0:
   NEXT i
80 FOR i=75 TO 180 STEP 2: PLOT i,70: DRAW 0,15:
   NEXT i
90 READ a,b,a1,b1,x,y,x0,y0,dx,dy,w1,w2,s1,s2
100 PRINT AT x,y;OVER 1;"D": GO SUB 380: GO TO
   130
110 IF a=a1 AND b=b1 THEN GO TO 140
120 PRINT AT 8,a1;INK 2,OVER 1;"A"; AT 19,b1;INK
   7,"B"; AT 20,b1;"C"
130 PRINT AT 8,INT a;INK 2,OVER 1;"A"; AT 19,b;
   INK 1;"B",AT 20,b;"C"
140 PRINT AT INT x,INT y,OVER 1,"D"; AT x0,y0;
   OVER 1;"D"
150 LET x0=INT x: LET y0=INT y: LET a1=INT a: LET
   b1=b
160 IF x=8 THEN GO TO 210
170 IF x=19 THEN GO TO 280
180 LET b=b+( INKEY$="8" AND b<26)-( INKEY$="5"
   AND b>5)
190 LET a=a+st*((a1< INT y AND a<20)-(a1> INT y
   AND a>11))
200 LET x=x+dx: LET y=y+dy: GO TO 110
210 IF INT y<> INT a THEN GO TO 250
220 BEEP .06,0
```

```
230 LET r=RND: LEt dy=(r>.6 AND a<19)/2-(R<.8 AND
    a>12)/2
240 LET x=x+1: LET dx=1: GO TO 110
250 BEEP .2,-20: LET w2=w2+1
260 IF w2>3 THEN LET s2=s2+1: LET w2=0: LET w1=0:
    IF s2>5 THEN GO TO 340
270 FOR t=1 TO 99: NEXT t: IET y=a: GO SUB 380:
    GO TO 220
280 IF INT y<>b THEN GO TO 310
290 BEEP .05,20
300 LET dy=(b<15)/2-(b>17)/2: LET dx=1: LET x=-1:
    GO TO 110
310 BEEP .2,-20: LET w1=w1+1
320 IF w1>3 THEN LET s1=s1+1: LET w1=0: LET w2=0:
    IF s1>5 THEN GO TO 340
330 FOR t=1 TO 99: NEXT t: LET y=b: GO SUB 380:
    GO TO 290
340 PRINT AT 14,3, INK 0; PAPER 7;
350 IF s2=6 THEN PRINT "Vyhral jsi zapas. Vyslede
    k: ";s1," ";s2: BEEP 1,-9
360 IF s1=6 THEN PRINT "Prohral jsi zapas. Vysled
    ek: ";s1,":";s2: BEEP 1,9
370 PRINT AT 16,4, FLASH 1,"Zkus hrat jeste jedno
    u!": PAUSE 0: RUN
380 PRINT AT 0,0;INK 2,PAPER 7,S1,"  ",BRIGHT 1,
    FN W (w1);" "
390 PRINT #0, AT 1,0;INK 1; PAPER 7,s2;"  ";BRIGH
    T 1; FN W (w2);" "
400 RETURN
410 DEF FN W (p) = 15*p-5*(p=3)
420 DATA 3,27,25,62,68,24,36,66,3,3,63,61,61,1,
    127,127,188,188
430 DATA 60,60,102,102,102,231,0,0,24,60,60,24,0,
    0
440 DATA 540,16,175,0,-47,87,-81,0,-47,-87,556,16
```

```
450 DATA -39,87,570,45,115,0,627,45,0,25,590,90,7
    5,0,627,90,0,-7
460 DATA 570,70,115,0,0,15,-115,0,0,-15,999,0
470 DATA 15,15,15,15,18,15,18,15,-1,0,0,0,0,0
```

Řádky 70 a 80 kreslí ve středu hrací plochy síť, na řádku 90 jsou proměnným přiřazeny počáteční hodnoty, čtené ze souboru DATA.

Proměnné A a B představují aktuální postavení "počítačového" závodníka i závodníka řízeného hráčem. Proměnné A1 a B1 jsou předešlá postavení obou závodníků.

Proměnné X a Y jsou aktuální souřadnice míčku, X0 a Y0 jsou předešlé. Proměnné DX a DY označují způsob pohybu míčku, jsou to přírůstky odpovídajících souřadnic po jednotlivých krocích. W1 a W2 označují počet vyhrávajících míčků ve hře pro počítač a hráče, S1 a S2 počet vyhraných her obou stran od počátku setu.

Vlastní tělo programu, jeho t.zv. hlavní smyčka, je na řádcích 110 až 200. POZOR! při vkládání řádku 100 a 120 až 140, všude tam, kde je v příkazech PRINT velké písmeno "A" až "D", je nutno stisknout příslušnou klávesu v grafickém modu, aby byly vyvolány z UDG příslušné definované znaky.

Aby došlo k přemístění kteřchokoli závodníka na novou pozici (jiné místo), je třeba nejprve vymazat jeho obraz v původní pozici (řádek 120) a nakreslit jeho obrázek na novou pozici (řádek 130). Podobně se přemísťuje i míček. V řádcích 160 a 170 se kontroluje, zda míček nevypadl ze hry přes některou z koncových čar. Pokud ano, následuje skok do odpovídajícího podprogramu kontrolujícího, zda je v tomto místě závodník. Pokud ano, má se za to, že míček byl odražen. V opačném případě bude protivníkovi přiznán další bod.

Po každém odražení míčku počítačem následuje výpočet nových parametrů jeho letu pomocí funkce RND (řádek 230). Při odražení míčku hráčem řízeným "ručně" je let míčku vypočten na základě aktuálního postavení závodníka (řádek 300). Po změně stavu

zápasu, pokud některá ze stran vyhraje podání, je na obrazovce vytištěn aktuální výsledek (stav skore). Přepočtem vyhraných míčků na tenisové body se zabývá funkce FN W (řádek 410), tisk výsledků obstarává podprogram od ř. 380.

Odražení míčku hráčem, případně dopad míčku mimo hrací plochu jsou signalizovány odpovídajícími zvukovými efekty, které má na starosti příkaz BEEP. Pokud kterákoli ze stran získala šestou hru, je tento stav zjištěn v řádcích 350 a 360 a odpovídající výsledek spolu s konečným výsledkem je vytištěn, hráč je pak pozván k dalšímu zápasu. Po ztrátě míčku a před následujícím servisem je nutná krátká přestávka. Tětp je docíleno pomocí zpožďovací smyčky, používající cyklus FOR....NEXT, na řádcích 270 a 330. Zdánlivě jednodušší by se jevilo použití příkazu PAUSE, ale připomeňme si, že vykonávání tohoto příkazu končí v okamžiku stisku libovolné klávesy. Je dost pravděpodobné, že hráč by v zápalu boje stiskl některou klávesu a tak nepřirozeně zkrátil přestávku před dalším podáním.

Ovládání dolního, "ručního" závodníka obstarává řádek 180, který pomocí funkce INKEY$ sleduje klávesnici a rozpoznává stisk některé z ovládacích kláves "5" nebo "8" (pochopitelně, že je zde možno vybrat i jiné klávesy).

Řízením "počítačového" závodníka se zabývá řádek 190. Jeho nové postavení je určeno na základě toho, na které jeho straně se nachází aktuální míček. Čím vyšší stupeň obližnosti ST, tím lépe reaguje "počítačový" hráč na pohyb míčku.

Uvedený program je pouze kostrou počítačové hry. Je možné jej značně vylepsit, vykreslením např. tribun s diváky, počítání skore hry přesně podle tenisových pravidel (převaha dvou míčků, která je potřebná k vítězství ve hře atd.).

A JEŠTĚ RŮZNÉ DROBNOSTI.....

Původně jsem se domníval, že popisem programu TENIS toto povídání zakončím, ale pak po diskusích s uživateli počítačů SPECTRUM, DELTA a DIDAKTIK GAMA jsem se odhodlal zařadit ještě několik krátkých programů, jako ukázky, co všechno se dá dělat jednoduchými prostředky. Doufám, že pro mnohé z nás budou inspirací a pobídkou k další tvorbě.

ČESKÁ ABECEDA

Pro ty, kteří nemají zájem pohrát si s převodem znaků počítače na znaky české abecedy, předkládám zde program ve strojovém kodu, který nadefinuje do oblasti UDG znaků česká písmena:

Q = ě	E = é	R = ř	T = ť	U = ů	I = í
P = Ř	A = á	S = š	D = d	F = č	G = ž
O = o	H = ý	J = ú	K = Š	L = Ú	C = č
B = ž	N = ň	M = D			

Pokud budeme chtít vypsat český znak, např. písmeno "ř", stiskneme CAPS SHIFT + GRAPHICS, pak klávesu "R" a zpět do normálních znaků se vrátíme stiskem klávesy "9" (t.j. GRAPHICS).

PROGRAM 56

```
 1 REM CESTINA
 2 DATA 65368,2
 3 DATA "0810380437C443C0010"
 4 DATA "2810E7C08102E7C0068"
 5 DATA "14081C2020201C00B4"
 6 DATA "0000506JC4444JC000B"
 7 DATA "0810384478403C0088"
 8 DATA "3C3C4240404237C00B8"
 9 DATA "3C7E04081020/E0074"
10 DATA "0810444444437C04385C"
11 DATA "0810003010103800A0"
12 DATA "0810444444444380060"
13 DATA "3C3C403C02423C0074"
14 DATA "0852424242423C009E"
15 DATA "3C7844424244780038"
16 DATA "2810784444444400C0"
17 DATA "0810384444444380054"
18 DATA "3C7C42427C4442003E"
19 DATA "2810384478403C00A8"
20 DATA "14081C2020202000H8"
21 DATA "2810384038047800064"
22 DATA "0A1410381010CC0092"
23 DATA "1000444444444380058"
24 DATA ""
```

Při zavádění tohoto programu do paměti počítače si opět pomůžeme programem č. 13, po spuštění a zdárném dokončení můžeme zadat příkaz NEW. Program v BASIC se vymaže, ale znaky uložené v oblasti UDG zůstanou zachovány. Nyní si můžeme vložit svůj program, ve kterém potřebujeme psát česky, nebo vytvořený znakový soubor nahrát na kazetu příkazem

SAVE "cestina"CODE 65368,168.

Zpět do počítače jej pak kdykoli přehrajeme pomocí

LOAD "cestina"CODE

a můžeme opět použít.

VÝPIS PROMĚNNÝCH

Při rozboru nějakého programu (dokonce někdy i vlastního) se nám docela snadno může stát, že zabloudíme v názvech proměnných. Pokud potřebujeme ještě nějakou proměnnou doplnit, tak, abychom zároveň nepřepsali jinou, v programu již použitou, musíme si udělat jejich seznam, což u delšího programu není nic jednoduchého. To už je jednodušší použít program ve strojovém kódu, který tuto otravnou práci obstará za nás.

Program 58 je relokalibilní, změnou adresy na řádku 9691 jej můžeme uložit na libovolné místo paměti.

PROGRAM 58

```
9690 REM vypis promennych
9691 DATA 60000,9691
9692 DATA "CD6B0DAF233C5C2A4B33"
9693 DATA "5C7EFE80C8E61FC6604B"
9694 DATA "D77E23E6E0FEE0281950"
9695 DATA "FE402820FE402825FE6F"
9696 DATA "C0282CFE80282B0105EB"
9697 DATA "0009E0DD718D63EEB42"
9698 DATA "D73EF3D701120018EFF9"
9699 DATA "3E24D74E23462318E611"
9700 DATA "7EE67FD7CB7E2328F745"
9701 DATA "18D83E24D73E28D74EB4"
9702 DATA "234623091E5ED4246C5B4"
9703 DATA "234E2346E5CD2B2DCDB1"
9704 DATA "E32D3E2CD7E1C110EDF0"
9705 DATA "E13E08D7E3291D718B105"
9706 DATA ""
```

Po uložení strojového programu do paměti počítače pomocí programu 13 (loader) si jeho funkci můžeme přezkoušet pomocí demonstračního programu.

PROGRAM 59

```
10 REM DEMO - vypis promennych
20 LET xmin=0: DIM e(3,10): DIM i$(2,2,4)
30 LET adr=60000
40 FOR n=xmin TO adr: NEXT n
50 LET a=5: LET b=a-2: LET c=b+10
```

Po jeho spuštění příkazem RUN se popsané proměnné uloží do paměti počítače, nyní zadáme RANDOMIZE USR adr a na obrazovce se objeví seznam:

```
xmin
e(3,10)
i$(2,2,4)
adr
n FOR NEXT
a
b
c
```

Ještě poznámka pro nepozorné, a pro ty, kteří tuto příručku čtou na přeskáčku (ostatní to již umí):
Program ve strojovém kodu uložíme na kazetu příkazem

```
SAVE "promenne"CODE 60000,126
```

a zpět do počítače:

```
CLEAR 59999
LOAD "promenne"CODE
```

Pokud jej chceme nahrát na jiné místo paměti, než byl uložen původně, nic nám nebrání použít následující sled příkazů:

```
CLEAR (adresa-1)
LOAD "promenne"CODE (adresa)
```

Spustíme jej pak RANDOMIZE USR (adresa uložení).

ELEGANTNÍ CLS.

V některých novějších profesionálních programech her se používá zajímavý způsob mazání obrazovky (například VENOM STRIKES BACK), na obrazovce se objeví osmibarevné políčko, které se posouvá a při tom maže obsah obrazovky. Jelikož s tento zajímavý efekt dá použít v libovolném jiném programu, předkládám jej dalším zájemcům.

Počáteční adresa podprogramu je 40 000, ale je možno ji libovolně upravit dle požadavků -- program je plně relokalibilní, díky použití proměnných GOTO1, GOTO2 a GOTO3. Pro zjednodušení provádění úprav programu (jeho adresace) jsem upustil od dosud používaného způsobu zavádění strojního kodu v hexadecimálním vyjádření pomocí programu 13.

Tento podprogram je možno použít i ke změně barvy obrazu, stačí přepsat obsah adresy (ADR+26), která uchovává kod barvy. Je-li jeho hodnota menší než 45, pak musíme na adresu (ADR+5) vepsat hodnotu 100 a na (ADR+9) zvětšíme obsah o tolik, o kolik byl změněn kod barvy. Připomínám ještě, že hodnota kodu barvy nemůže být větší než 71.

Pokud je obsah adresy (ADR+26) větší než 45, je nutno postupovat podobně a na adresu (ADR+5) vložit hodnotu 118. Proč? To již nechávám k luštění hloubavějším povahám, jistě na to brzy přijdete a tímto se zdokonalíte v používání strojového kodu.

Na adrese (ADR+35) je uchováván počet řádků v běžícím políčku, na (ADR+44) počet řádků, určených ke smazání.

Před zavedením strojového programu do paměti počítače tyto hodnoty neupravujte, jelikož zaváděcí program by pak hlásil chybu v datech. Úpravy můžeme provést až v hotovém, zavedeném strojovém programu pomocí příkazů POKE.

PROGRAM 60

```
10 REM eleganlni CLS
20 LET adr=40000: LET s=0
30 LET golo1=adr+13: LET golo2=adr+41: LET golo3
   =adr+20
40 FOR i=0 TO 75: READ a
50 POKE (adr+i),a
60 LET s=s+a: NEXT i
70 IF s<>9212 THEN PRINT "Chyba v datech!": STOP
80 DATA 30,248,205,golo1-256*INT (golo1/256)
90 DATA INT (golo1/256),118,28,123,254,120
100 DATA 32,246,201,213,245,22,0,213,62,71,205
110 DATA golo2-256*INT (golo2/256),INT (golo2/256
    )
120 DATA 29,61,254,63,194
130 DATA golo3-256*INT (golo3/256),INT (golo3/256
    )
140 DATA 209,29,20,122,254,24,32,235,241,209,201
150 DATA 8,123,254,32,48,27,122,254,24,48,22,229
160 DATA 33,0,88,203,58,203,29,203,58,203,29
170 DATA 203,58,203,29,25,87,8,119,225,201,8,201
```

MAZÁNÍ ŘÁDKU V BASIC

Program slouží k vymazání libovolného počtu po sobě následujících řádků v BASICovém programu. Spustíme jej příkazem GO TO 9990 a postupně vložíme číslo prvého řádku, který se má smazat (viz proměnná X) a číslo prvého řádku, který již má být zachován (proměnná Y).

Program si upraví příslušné údaje v souboru DATA a načte je do paměti jako strojový kod. Po proběhnutí zaváděcího programu se automaticky spustí strojový program (na řádku 9998) a vrátí se zpět do BASIC s hlášením OK.

Nyní již stačí ručně vymazat prvý určený řádek (t.j. s číslem vloženým do proměnné X) a všechny ostatní požadované řádky programu se již vymažou automaticky.

Tento program lze použít např. pro vymazání zaváděcích programů (LOADERů) strojového kodu, kontrolních programů a dokonce i sebe samého (nakonec).

PROGRAM 61

```
9990 INPUT x,y
9991 LET xa=INT (x/256)
9992 LET xb=x-xa*256
9993 LET ya=INT (y/256)
9994 LET yb=y-ya*256
9995 RESTORE 9999
9996 FOR a=65400 TO 65452
9997 READ b: POKE a,b: NEXT a
9998 RANDOMIZE USR 65400
9999 DATA 1,xb,xa,42,83,92,205,163,255,34,175,
     92,1,yb,ya,43,205,163,255,58,77,42,176,92,
     237,161,237,161,121,149,79,120,156,119,43,
     113,201,35,94,35,86,25,35,126,184,35,32,
     245,126,185,32,241,201
```

PROGRAMOVÁ LUPA.

Tento program je určen pro zvídavější povahy, pro ty, kteří chtějí vědět, jakým způsobem je vlastně program psaný v jazyku BASIC uložen do paměti počítače. Po vložení tohoto programu (ať již z klávesnice, nebo z kazety příkazem LOAD) zavedeme do počítače program, který chceme rozlušlit, pomocí příkazu MERGE.. Nyní již stačí jen zavelet GO TO 9000 a na obrazovce se nám objeví počáteční adresa programu v BASIC, bajt uložený na této adrese a jeho význam. Po stisku kterékoli další klávesy (mimo N) vytiskne program další adresu paměti s příslušnými údaji.

Program se nám může hodit i v těch případech, kdy je program v BASIC nějakým způsobem zatajený, takže normálně je nečitelný. Jeho pomocí můžeme ulajené programy nejenom rozluštit, ale i upravit tak, aby byly čitelné (vypuštěním dodatečně vložených řídících kodů).

PROGRAM 62

```
9000 REM Programova lupa
9010 CLS: LET s=PEEK 23635+PEEK 23636*256-1: GO
     TO 9210
9020 LET p=PEEK s
9030 IF p=13 THEN GO TO 9200
9040 IF p=14 THEN GO TO 9300
9050 IF p<10 THEN PRINT s;"    ";"    ";
9060 IF p>9 AND p<100 THEN PRINT s;"    ";p;
     "    ";
9070 IF p>99 THEN PRINT s;"    ";p;"    ";
9080 IF p>31 THEN PRINT PAPER 6;CHR$ p
9090 IF p<32 THEN PRINT
9100 LET s=s+1: GO TO 9020
9200 PRINT s;"    ";p;"    ";PAPER 4;"konec
     radku"
9205 PRINT
9210 LET s=s+1: LET p=PEEK s*256+PEEK (s+1)
9220 PRINT s;"    ";PAPER 4;"Radek c.:  ";p;
     "   "
9230 LET s=s+2: LET p=PEEK s+PEEK (s+1)*256
9240 PRINT s;"    ";PAPER 6;"Pocet bajtu = ";p
9250 LET s=s+1: GO TO 9100
9300 PRINT s;"    ";p;"    ";PAPER 4;"Oznaceni
     poctu: "
9310 PRINT s+1;"    ";
9320 FOR f=s+1 TO s+5
9330 PRINT PEEK f;"  ,  ";
9340 NEXT f
9350 PRINT: LET s=s+5: GO TO 9100
```

MIKROCOPY

Naprostá většina programů, které vlastní uživatelé všech typů mikropočítačů, alespoň u nás, jsou nikoli legálně zakoupené, ale získané výměnou od známých. Víme dobře, že např. firemní programy, nebo vůbec programy psané ve strojovém kodu, se nahrávají z počítače na kazetu jen velmi obtížně, pokud se vůbec nahrát dají. Obvyklé firemní programy se po zavedení do počítače okamžitě spustí a žádným nám známým jednoduchým způsobem nejdou zastavit, abychom si je mohli přehrát na novou kazetu.

Člověk je ale tvor koumavý a proti těmto firemním "finesám" vymyslel protiopatření - t.zv. kopírovací programy, zkráceně "kopíráky". Jak takový kopírovací program pracuje?

Je-li zaveden do počítače a spuštěn, přijímá další nahrávaný program již nikoli jako program, ale jako data - sled bajtů, která si ukládá do paměti. Po nahrání do počítače pak snadno můžeme tento sled bajtů nahrát zpět na další kazetu a tím si vytvořit kopii libovolného programu.

Každý z vás již určitě má ve své sbírce nějaký ten "kopírák", my si zde uvedeme ještě jeden velice jednoduchý, který je schopen okopírovat i velmi dlouhé programy - až 48945 bajtů, díky tomu, že sám nezabírá žádné místo v operační paměti počítače. Tělo skutečnosti bylo docíleno jednoduchým trikem - uložíme jej do obrazové paměti od adresy 16416.

Také jeho obsluha je velmi jednoduchá - po zavedení do počítače jej spustíme příkazem RUN a program je připraven k nahrávání kopírovaného programu. Po jeho nahrání se sám zastaví, stiskem libovolné klávesy se přepne do stavu "SAVE" a uložený program přehraje do magnetofonu. Po opětovném stisku některé klávesy je opět připraven přijmout další program, atd.

Většina firemních programů se skládá z několika částí - zaváděcí BASIC, SCREEN$ (obraz), CODE (program ve strojovém kodu). Každou z těchto částí tedy nahrajeme do kopírovacího programu samostatně a ihned přehrajeme na kazetu.

A nyní samotný program:

PROGRAM 63

```
10 FOR n=1 TO 173
20 READ a: POKE 16415+n,a: NEXT n
30 RANDOMIZE USR 16416
40 DATA 17,5,1,33,201,1,205,181,3,49,30,64,62
50 DATA 0,221,33,206,64,17,255,255,55,205,86,5
60 DATA 243,124,254,0,40,5,195,32,64,62,0,33,206
70 DATA 64,190,40,31,183,33,255,255,237,82,43
80 DATA 43,84,93,221,33,207,64,205,184,64,58,206
90 DATA 64,55,205,194,4,205,184,64,195,32,64,62
100 DATA 0,221,33,224,64,17,255,255,55,205,86,5
110 DATA 243,124,254,0,40,4,195,32,64,183,33,255
120 DATA 255,237,82,43,43,235,122,50,0,64,123,50
130 DATA 1,64,205,184,64,62,0,221,33,207,64,17,17
140 DATA 0,55,205,194,4,118,58,0,64,87,58,1,64,95
150 DATA 221,33,225,64,62,255,0,205,194,4,205,184
160 DATA 64,195,32,64,62,0,219,254,254,190,200
170 DATA 254,189,200,254,187,200,254,183,200,254
180 DATA 175,200,24,235
```

KRESLENÍ

Program je obdobou t.zv. "MAGICKÉ TABULKY" a slouží ke kreslení jednoduchých obrázků na obrazovku. Po vložení do počítače se spustí příkazem RUN a po stisku libovolné klávesy (mimo BREAK, pochopitelně) se uprostřed objeví blikající bod. Tento bod můžeme posouvat pomocí kurzorových kláves po celé ploše obrazovky, po stisku klávesy "1" přestane blikat a zanechává za sebou při svém pohybu stopu (čáru).

Stiskem klávesy "0" se opět rozbliká a je možno jej přesunout na jiné místo obrazovky, případně pohybem po již nakreslené čáře tuto smazat.

Do hotového obrázku můžeme ještě vepsat libovolný text - stiskem klávesy "T" se program přepne do textového modu a zeptá se nás na text, který chceme vložit. Po odeslání klávesou ENTER se zeptá na souřadnice X a Y, kde má být umístěn počátek textu (jako PRINT AT). Po všech odpovědích vytiskne vložený text na určené místo, opět se vrátí do grafického modu a očekává další příkazy. Nyní můžeme buď pokračovat v kreslení (např. k nápisu doplnit háčky a čárky, pokud nemáme v počítači zavedenu českou abecedu), nebo klávesou "S" nakreslený obrázek nahrát na kazetu (SAVE SCREEN$).

Pokud máme k dispozici tiskárnu, můžeme ještě do programu vložit řádek:

```
39 IF a$="c" THEN COPY
```

Většina tiskáren (mimo ZX PRINTER a SEIKOSHA GP 50 S) na příkaz COPY nepracuje, pak místo něj vložíme RANDOMIZE USR XXXXX, zde dosadíme adresu funkce COPY dané tiskárny. Tiskárna BT 100 však umí vytisknout obsah obrazovky v různých velikostech, takže zde máme k dispozici další volbu: buď nastavit jednu velikost a tu stále používat, nebo do programu doplnit další řádky:

```
 39 IF a$="c" THEN LET v=0: GO SUB 110
  .          .          .          .
  .          .          .          .
110 INPUT "Velikost 1 - 2 - 3 ? ";v
120 IF v=1 THEN RANDOMIZE USR XXXXX
130 IF v=2 THEN RANDOMIZE USR YYYYY
140 IF v=3 THEN RANDOMIZE USR ZZZZZ
150 RETURN
```

Po dokončení tisku, nebo pokud nezvolíme žádnou z nabídnutých možností, vrátí se program opět do grafického modu a můžeme pokračovat.

Ale nyní konečně vlastní program:

PROGRAM 64

```
 1 REM KRESLENI
 5 GO TO 20
10 POKE 23659,1: POKE 23734,111: POKE 23736,20:
   SAVE "Vobrazek"SCREEN$
20 PAUSE 0: CLS: LET x=VAL "127": LET y=VAL
   "87": LET m$="0"
30 LET l=VAL "63486": LET r=VAL "61438"
34 LET m=VAL m$
35 PLOT INVERSE 0;x,y: LET a$=INKEY$: IF a$="s"
   THEN GO TO 10
36 IF a$="0" OR a$="1" THEN LET m$=a$
37 IF m$="0" THEN PLOT INVERSE 1;x,y
38 IF a$="t" THEN GO SUB 100
40 LET x=x+(x<255)*(NOT INT (IN r/4)-2*INT (IN r
   /8))-(x>0)*(NOT INT (IN l/16)-2*INT (IN l/32)
   )
60 LET y=y+(y<175)*(NOT INT (IN r/8)-2*INT (IN r
   /16))-(y>0)*(NOT INT (IN r/16)-2*INT (IN r/32
   ))
70 IF INKEY$="s" THEN GO TO 10
80 PLOT INVERSE m;x,y
90 GO TO 35
100 INPUT "TEXT: ";t$:INPUT "X? ";a,"Y? ";b:
   PRINT AT a,b;t$: RETURN
```

NĚKOLIK GRAFICKÝCH PODPRORAMŮ.

Počet přímek k daným bodům.

Program očekává zadání libovolného počtu bodů, minimálně však 2, vymyslí si pro tyto body jejich umístění na ploše obrazovky pomocí funkce RND (řádky 30 až 60). Vykreslí zadaný počet libovolně rozmístěných bodů každý s každým navzájem propojí přímkami. Proto není vhodné zadávat příliš velký počet bodů, vznikla by nepřehledná změť čar.

Nakonec oznámí, kolik přímek je zapotřebí k propojení všech těchto bodů. Program se tedy dá použít i k demonstračním účelům v matematice.

Ještě poznámka - jak je vidět z řádků 40 a 50, je i stejný počet zadaných bodů rozmístěn pokaždé jinak, vznikne tedy pokaždé i jiný obrazec.

PROGRAM 65

```
  5 REM Pocet primek k danym bodum
 10 CLS: INPUT "POCET BODU? ";a
 20 DIM x(a): DIM y(a): LET k=0
 30 FOR f=1 TO a
 40 LET x(f)=RND * 255
 50 LET y(f)=RND * 175
 60 NEXT f
 70 FOR f=1 TO a
 80 LET x1=x(f): LET y1=y(f)
 90 FOR g=f+1 TO a
100 PLOT x1,y1: DRAW (x(g)-x1),(y(g)-y1)
110 LET k=k+1: NEXT g: NEXT f
120 PRINT #0;AT 0,0;a;" bodu propoji ";k;
    " primek."
130 PAUSE 0: GO TO 10
```

Koberec.

Program demonstruje zajímavé použití příkazů DRAW, vykreslí "kobercový" vzor na ploše obrazovky. Dá se využít např. jako titulní obrázek k některému programu, pokud přes hotový obrázek napíšeme příslušné titulky pomocí příkazu PRINT AT.

PROGRAM 66

```
150 REM Koberec
160 FOR f=0 TO 255 STEP 5
170 PLOT f,0: DRAW 0,175
180 NEXT f
190 FOR f=0 TO 255 STEP 5
200 PLOT 0,0: DRAW f,175: NEXT f
210 FOR f=0 TO 255 STEP 5
220 PLOT 255,175: DRAW -f,175: NEXT f
230 FOR f=0 TO 255 STEP 5
240 PLOT 255,0: DRAW -f,175: NEXT f
250 FOR f=0 TO 255 STEP 5
260 PLOT 0,175: DRAW f,-175: NEXT f
270 PAUSE 0
```

Prostorová kresba.

Tento program se používá u celé řady počítačů k demonstraci jejich grafických možností, např. IQ 151 ve spojení se souřadnicovým zapisovačem MINIGRAF a pod.

Spolu se dvěma předešlými programy jsem jej převzal z počítače ZX 81, upraveného na jemnou grafiku (mimochodem - pomocí pouze dvou přidaných diod).

PROGRAM 67

```
300 REM Prostorova kresba
310 LET xa=170: LET ya=0: LET xb=170: LET yb=85
320 LET xc=246.6393: LET yc=43.2
```

```
330 GO SUB 400
340 LET xa=xc: LET ya=126.8: GO SUB 400
350 LET xc=170: LET yc=170: GO SUB 400
360 LET xa=94.754: LET ya=126.8: GO SUB 400
370 LET xc=xa: LET yc=43.2: GO SUB 400
380 LET xa=170: LET ya=0: GO SUB 400
390 PAUSE 0: STOP
400 REM podprogram kresby
410 LET x1=xa: LET y1=ya: LET x2=xb: LET y2=yb
420 LET x3=xc: LET y3=yc
430 LET d=0.1: LET i=20
440 FOR f=1 TO i
450 PLOT x1,y1: DRAW x2-x1,y2-y1
460 DRAW x3-x2,y3-y2
470 LET x=(x2-x1)*d+x1
480 LET y=(y2-y1)*d+y1
490 LET x2=(x3-x2)*d+x2
500 LET y2=(y3-y2)*d+y2
510 LET x3=(x1-x3)*d+x3
520 LET y3=(y1-y3)*d+y3
530 LET x1=x: LET y1=y: NEXT f
540 RETURN
```

Procvičování matematiky.

Když se náš počítač jmenuje DIDAKTIK GAMA, tak si tedy uvedeme ještě široce použitelný DIDAKTICKÝ program.

Na 2. stupni základní školy má značný počet žáků potíže s numerickým výpočtem, hlavně při násobení a dělení. Nezáživnost učiva a monotonní práci zkoušejícího (nejenom učitele, ale i rodičů doma) překlene náš neúnavný pomocník - počítač. Uvedený program je použitelný od 3. až k nejvyšším ročníkům ZŠ.

Program generuje celá náhodná čísla v rozsahu [0 - 10] a úkolem zkoušeného je najít jejich součin. V dalším řádku následuje dělení beze zbytku celého čísla v rozsahu [0 - 100] celým číslem [1 - 10].

Další úlohu nám zadá teprve až po správném vyřešení předcházející úlohy. Pokud se žák dopustí chyby, program vypíše na obrazovku chybové hlášení a ozve se akustickým signálem. Teprve pak je možno pokračovat dále.

Celkem je zadáno 40 příkladů a čas jejich řešení je měřen, čímž je možno srovnávat úspěšnost různých žáků, nebo postup zlepšování schopností jednoho žáka.

Program po svém spuštění příkazem RUN zadá příklad, pomocí klávesnice vložíme správnou odpověď a odešleme ji klávesou ENTER. Pokud byla odpověď správná, počítač nám předloží další úlohu. Nesprávnou odpověď opraví, jak již bylo uvedeno, a pokračuje nerušeně dále. Pokud jsme při vkládání odpovědi stiskli omylem nesprávnou klávesu, můžeme se opravit (před odesláním ENTER) klávesou DELETE.

PROGRAM 68

```
10 REM Nasobeni, deleni
20 RANDOMIZE
30 LET w$="Hodnoceni: "
40 LET c$="CHYBA - Spravne ma byt: "
50 LET d=0
60 FOR n=1 to 20
70 LET x=INT (RND * 11)
80 LET y=INT (RND * 11)
90 PRINT x;"*";y;"=";
100 INPUT z
110 PRINT z
120 IF n>2 THEN GO TO 140
130 LET a= FN T ()
```

```
140 IF x*y<>z THEN GO SUB 430
150 LET p=INT (RND * 11)
160 LET r=INT (RND * 10)+1
170 LET s=p*r
180 PRINT s;":";r;"=";
190 INPUT t
200 PRINT t
210 IF t<>s/r THEN GO SUB 470
220 PRINT: NEXT n: PRINT
230 LET b= FN T ()
240 PRINT
250 LET c=b-a
260 LET m=INT (c/60)
270 LET v=c-60*m
280 PRINT "Minul: ";m;"   Sekund: ";v
290 PRINT: PRINT
300 PRINT "Zadal jsem Ti 40 uloh, z toho"
310 PRINT: PRINT "jsi spatne odpovedel ";d
320 PRINT: PRINT
330 PRINT w$;
340 IF d<=2 THEN PRINT "1"
350 IF d=3 OR d=4 THEN PRINT "2"
360 IF d=5 OR d=6 THEN PRINT "3"
370 IF d=7 OR d=8 THEN PRINT "4"
380 IF d>8 THEN PRINT "5"
390 PRINT AT 21,0;" OPAKOVAT - O,    KONEC - K"
400 IF INKEY$="k" THEN STOP
410 IF INKEY$="o" THEN RUN
420 GO TO 400
430 PAUSE 55: BEEP 1,30
440 PRINT c$;x;"*";y;"=";x*y
450 LET d=d+1
460 RETURN
470 PAUSE 55: BEEP 1,30
480 PRINT c$;s;":";r;"=";s/r
```

```
490 LET d=d+1
500 RETURN
999 DEF FN T ()=(65536*PEEK 23674+256*PEEK 23673
   +PEEK 23672)/50
```

Zavedle si pořádek v knihovně pomocí programu KARTOTÉKA.

Program slouží k ukládání a úschově dat, v této verzi je určen k vedení přehledu o obsahu knihovničky. Po změně názvů ukládaných dat může sloužit i k jiným účelům, podle přání, např. k registraci článků v časopisech, adresy a telefonní čísla známých a podobně. V jedné z verzí (po drobných úpravách) jsem jej použil i jako "kuchařku", kde vstupními daty jsou údaje o obsahu spíže a výstupním údajem je návrh, co se z toho dá uvařit, ale to je spíše kuriozita.

Celá obsluha programu se děje pomocí t.zv. MENU, to je seznam nabídek programových možností, žádaný druh činnosti volíme stiskem tlačítka s příslušným číslem.

Při vkládání dat si program jednotlivé údaje vyžádá sám, takže stačí pouze sledovat výpisy na obrazovce a řídit se podle nich. Například po žádosti "DRUH?" vypíšeme buď SCI-FI, DET (jako detektivka), CESTOPIS, ROMAN atd. "NAZEV KNIHY" - zde je vkládaná informace jasná, rovněž tak "AUTOR".

Po žádosti počítače o "STRUCNY OBSAH" - můžeme vepsat buď "DOBRY", "NUDA", "POPIS CESTY MIKLUCHO-MAKLAJE", nebo dle vlastní úvahy cokoli podobného.

Celkem můžeme uložit údaje až o 250 knihách, pokud bychom omezili délky jednotlivých vstupních dat, dalo by se jich uložit i více - viz řádky 10 a 20.

PROGRAM 69

```
 1 REM KARTOTEKA
 5 POKE VAL "23609",VAL "50": POKE VAL "23658",
   VAL "8"
10 DIM v(SGN PI): LET z=VAL "250": LET m=VAL
   "1000": LET n=SGN PI
20 DIM a$(z,CODE " "): DIM b$(z,CODE " "): DIM
   c$(z,VAL "10"): DIM d$(z,VAL "64"): LET g=
   VAL "30": LET p=VAL "23617": LET r=CODE
   "FORMAT ": LET o=CODE "OUT "
30 CLS: PRINT BRIGHT 1;INVERSE 1;"    K N I H O
   V N A      ";v(1);"   ZAZNAMU    "
40 PRINT '''" 1 - VKLADANI UDAJU"''" 2 - VYBER
   PODLE NAZVU"''" 3 - VYBER PODLE AUTORA"''" 4
   - VYPIS VSEHO"''" 5 - SAVE DATA"''" 6 - LOAD
   DATA"''" 7 - SAVE ALL"''''"UKLADANI UKONCIT
   STISKEM";INVERSE 1;BRIGHT 1;" ENTER "
50 PAUSE 0: LET Q$=INKEY$: IF Q$<"1" OR Q$>"7"
   THEN GO TO VAL "50"
60 LET a=VAL Q$: CLS: GO TO a*CODE "d"
100 FOR s=n TO z: IF a$(s,SGN PI)<>" " THEN GO TO
   VAL "110"
105 NEXT s: GO TO g
110 POKE p,r: INPUT "DRUH: ";a$(s): PRINT a$(s):
   IF a$(s, SGN PI)="" THEN GO TO g
120 POKE p,r: INPUT "NAZEV KNIHY (max 1 radek) ";
   b$(s): PRINT b$(s)
130 POKE p,r: INPUT "AUTOR (max 10 znaku) ";c$(s)
   : PRINT c$(s)
140 POKE p,r: INPUT "STRUCNY OBSAH (max 2 radky)"
   ;d$(s): PRINT d$(s)
150 PRINT #0;AT 0,0;BRIGHT 1;"          JE TO OK?
      A=ANO, N=NE                ": PAUSE 0
160 IF INKEY$="N" THEN CLS: GO TO CODE "d"
```

```
170 IF INKEY$<>"A" THEN GO TO VAL "160"
180 IF s>v(SGN PI) THEN LET v(SGN PI)=s
190 LET n=s: CLS: GO TO CODE "d"
200 DIM x$(SGN PI, CODE " "): POKE p,o: INPUT
    "NAZEV KNIHY: ";x$(SGN PI)
210 FOR x=SGN PI TO v(SGN PI): IF x$(SGN PI)=b$
    (x) THEN GO SUB m
220 NEXT x: GO TO (m+m)
300 DIM x$(SGN PI,VAL "10"): POKE p,o: INPUT
    "AUTOR: ";x$(SGN PI)
310 FOR x=SGN PI TO v(SGN PI): IF x$(SGN PI)=c$
    (x) THEN GO SUB m
320 NEXT x: GO TO (m+m)
400 CLS: PRINT BRIGHT 1; INVERSE 1;" SEZNAM KNIH
    ";v(SGN PI)
410 FOR x=1 TO v(SGN PI): IF a$(SGN PI)<>" " THEN
    GO SUB m
420 PAUSE g: IF INKEY$<>"" THEN GO TO g
430 NEXT x: GO TO (m+m)
500 LET y=VAL "23736": LET t=CODE "ASN ": CLS:
    SAVE "POCET"DATA v(): POKE y,t: SAVE "DRUH"
    DATA a$(): POKE y,t: SAVE "KNIHA"DATA b$():
    POKE y,t: SAVE "AUTOR"DATA c$(): POKE y,t:
    SAVE "OBSAH"DATA d$(): GO TO g
600 CLS: LOAD ""DATA v(): LOAD ""DATA a$(): LOAD
    ""DATA b$(): LOAD ""DATA c$(): LOAD ""DATA d$
    (): PAUSE g: GO TO g
700 CLS: SAVE "SEZNAM ALL"LINE 30: GO TO g
1000 PRINT 'a$(x)'b$(x)'c$(x)'d$(x)'"----------
    ----------": RETURN
2000 PRINT BRIGHT 1; INVERSE 1;"          KONEC
    SEZNAMU                    "
2010 PRINT #NOT PI; AT NOT PI,NOT PI; BRIGHT SGN
    PI;" ZA DUVERU DEKUJE HELLSOFT c 1985 "
2020 PAUSE 0: GO TO g
```

Několik matematických podprogramů

Dělení čísel s velkou přesností.

Didaktik GAMA nám bohužel počítá pouze na max. 7 desetinných míst, někdy by se mohlo hodit určení výsledku na více míst - pak si pomůžeme následujícím prográmkem.

PROGRAM 70

```
10 PRINT TAB 8;"DELENI DVOU CISEL"'TAB 5;"SE ZVY
   SENOU PRESNOSTI"
20 PRINT TAB 13;"A/B=X"
25 PRINT "----------------------------------------"
30 INPUT "VLOZ A: ";a,"VLOZ B: ";b: PRINT "A=";
   a,"B=";b
40 LET x=INT (a/b): PRINT "X= ";x,".";
50 LET a=(a-(x*b))*10: Let x=INT (a/b)
60 PRINT x;: GO TO 50
```

Jako téměř každý program, i tento se dá upravit podle potřeby. Zde nám vypisuje desetinná místa až do (téměř) nekonečna, pokud požadujeme jen určitý počet míst, stačí přidat řádky:

```
35 INPUT "POCET MIST: ";p
45 FOR n=1 TO p
60 PRINT x;: NEXT n: PRINT : GO TO 25
```

Určení všech dělitelů přirozených čísel.

Také tento program patří do série pomocných matematických programků, všechny se dají použít i jako podprogram ve větším, který si píšeme sami. Stačí upravit čísla řádků, nakonec připsat RETURN a volat jej příkazem GO SUB.

PROGRAM 71

```
10 PRINT "    URCENI VSECH DELITELU"'"
   PRIROZENYCH CISEL"
20 PRINT "--------------------------------------"
30 INPUT "VLOZ N= ";n: PRINT : LET a=2
40 IF a-n=0 THEN GO TO 130
50 LET j=a
60 LET d=n/j
70 IF d-INT d=0 THEN GO TO 110
80 LET j=j+1
90 IF d-j<0 THEN GO TO 130
100 GO TO 60
110 PRINT j,d
120 GO TO 80
130 PRINT : PRINT "---------- KONEC ----------
   ---"
140 PAUSE 0: RUN
```

Faktoriál velkých čísel.

Obvykle se faktoriály dají počítat max. z čísla 69, pro vyšší čísla je možno použít následující programek:

PROGRAM 72

```
10 PRINT "     * FACTORIAL CISLA N *"
20 PRINT "-------------------------------------"
30 INPUT "N= ";n: LET a=n: LET r=0
40 FOR i=1 TO n
50 LET r= LN a/ LN 10+r
60 LET a=a-1: NEXT i
70 LET e=INT r
80 LET m=10↑(r-e)
90 PRINT "FACTORIAL ";n;"! = ";m;"E";e
100 PRINT: GO TO 20
```

Rozklad čísla na prvočinitele.

PROGRAM 73

```
10 PRINT "*ROZKLAD CISLA NA PRVOCINITELE*"
15 PRINT "-------------------------------------"
20 INPUT "VLOZ CISLO K ROZKLADU, N= ";n
25 IF n<=1 OR (INT n=n)<0 THEN GO TO 20
30 PRINT "Prvocinitele cisla N jsou:"
35 LET d=2: LET b=-1: LET c=2
40 LET a=n/d
45 IF a-INT a=0 THEN GO TO 70
50 LET c=c-1
55 IF c<0 THEN GO TO 65
60 LET d=d*2-1: GO TO 40
65 LET d=d+b+3: GO TO 40
70 LET n=n/d
75 PRINT d;",  ";
80 IF n>1 THEN GO TO 40
85 PRINT : GO TO 15
```

Převod desetinného čísla na zlomek.

PROGRAM 74

```
10 PRINT " * PREVOD CISLA N NA ZLOMEK *"
15 PRINT "-------------------------------------"
```

```
20 INPUT "VLOZ N= ";n,"MAX. ODCHYLKA= ";r
25 LET a=n+1: LET c=INT a
30 LET d=n-c: LET e=1
35 LET f=1: LET g=0
40 IF ABS (c/e-n)<r THEN GO TO 75
45 LET b=1/d: LET a=b+1
50 LET d=b-INT a: LET h=b-d
55 LET x=c: LET c=c*h+f
60 LET f=x: LET x=e
65 LET e=e*h+g: LET g=x
70 GO TO 40
75 LET c= ABS c: LET e= ABS e
80 PRINT "CISLO ";n;" JAKO ZLOMEK JE:"
85 PRINT TAB 12;c;"/";e
90 PRINT: GO TO 15
```

Největší společný dělitel, nejmenší společný násobek.

PROGRAM 75

```
10 PRINT "NEJVETSI SPOLECNY DELITEL","NEJMENSI
   SPOLECNY NASOBEK."
20 PRINT "-----------------------------------------"
30 INPUT "Vyssi cislo: ";n1,"Nizsi cislo: ";n2
40 PRINT: PRINT "Zadal jsi cisla ";n1;" a ";n2
50 LET n=n1*n2
60 LET a=n1/n2
70 LET b=n1-n2*INT a
80 IF b=0 THEN GO TO 100
90 LET n1=n2: LET n2=b: GO TO 60
100 LET n1=n/n2
110 PRINT : PRINT "Jejich NSD je ",n2
120 PRINT : PRINT "Jejich NSN je ";n1
130 PRINT : PRINT "-----------------------------------"
140 PRINT : PRINT "   Dalsi zadani? A=ano, N=ne."
```

```
150 IF INKEY$="a" THEN RUN
160 IF INKEY$<>"n" THEN GO TO 150
170 CLS: PRINT AT 10,11; FLASH 1,"TAK AHOJ!"
180 STOP  REM - nebo RETURN do hlavniho prgm.
```

Vyhledávání prvočísel.

PROGRAM 76

```
 10 PRINT "**********************************"
 12 PRINT "*    VYHLEDAVANI PRVOCISEL       *"
 14 PRINT "**********************************"
 16 PRINT  PRINT "Rozsah hledani od 1 do ",
 18 INPUT m: PRINT m
 20 PRINT  PRINT "------------------------------"
 30 DIM z(m)  FOR x=1 TO m
 40 LET z(x)=x  NEXT x: LET x=2
 50 LET p=x
 60 FOR x=2*p TO m STEP p
 70 LET z(x)=0: NEXT x
 80 FOR x=p+1 TO m/2
 90 IF z(x)>0 THEN GO TO 50
100 NEXT x
110 FOR x=1 TO m
120 IF z(x)>0 THEN PRINT x,
130 NEXT x
```

ČASOVKA.

Do výpočetního střediska postupně přicházejí dosažené výsledky závodníků. Tyto údaje jsou zpracovány a tiskne se tabulka pořadí závodníků. Po zadání dalšího výsledku se tabulka aktualizuje. Zadává se startovní číslo závodníka a jeho dosažený výkon. Pokud je požadováno vyhodnocení nc podle času, ale podle počtu dosažených bodů (střelba, vrh a pod.), upravíme řádek 100

```
100 IF a(k+1)<a(k) THEN GO TO 145
```

PROGRAM 77

```
  5 REM "CASOVKA"
 10 DIM a(50): DIM l(50): LET z=0
 20 LET z=z+1: INPUT "ZAVODNIK c.: ";l(z)
 30 IF l(z)=0 THEN GO TO 20
 40 PRINT AT 10,1,l(z)," - ";
 50 INPUT "CAS: ";a(z)
 60 IF a(z)=0 THEN GO TO 50
 70 PRINT a(z)
 80 IF z=1 THEN GO TO 160
 90 LET c=0: FOR k=1 TO z-1
100 IF a(k+1)>=a(k) THEN GO TO 145
110 LET p=a(k): LET a(k)=a(k+1)
120 LET a(k+1)=p: LET r=l(k)
130 LET l(k)=l(k+1): LET l(k+1)=r
140 LET c=c+1
145 NEXT k
150 IF c<>0 THEN GO TO 90
160 CLS: PRINT "PORADI   ZAVODNIK   CAS"
170 PRINT "-------------------------------"
180 FOR i=1 TO z
190 PRINT TAB 3;i; TAB 12;l(i), TAB 20;a(i)
200 PRINT "-------------------------------"
210 NEXT i: PRINT "*******************************"
220 PAUSE 0: CLS: GO TO 20
```



```
150 IF INKEY$="a" THEN RUN
160 IF INKEY$<>"n" THEN GO TO 150
```

Pořadí 1.

Po skončení závodu se podle záznamů rozhodčích zadá celkový počet závodníků, pak dle startovních čísel se vkládají dosažené výkony (čas). Pokud je požadováno hodnocení dle počtu dosažených bodů, změníme řádek

```
80 IF a(k+1)<a(k) THEN GO TO 130
```

Po vložení všech informací program vytiskne tabulku výsledného pořadí závodníků. Tento, jakož i předešlý a následující program je možno případně doplnit o příkazy tisku na tiskárně a získat tak písemné vyhodnocení.

PROGRAM 78

```
 5 REM "PORADI 1"
10 INPUT "Pocet zavodniku? ";z
20 DIM a(z): DIM l(z)
30 PRINT "ZAVODNIK","CAS"
40 FOR i=1 TO z: PRINT i,
50 INPUT a(i): PRINT a(i)
60 LET l(i)=i: NEXT i
70 LET c=0: FOR k=1 TO z-1
80 IF a(k+1)>=a(k) THEN GO TO 130
90 LET p=a(k): LET a(k)=a(k+1)
100 LET a(k+1)=p: LET r=l(k)
110 LET l(k)=l(k+1): LET l(k+1)=r
120 LET c=c+1
130 NEXT k
140 IF c<>0 THEN GO TO 70
150 CLS: PRINT "PORADI   ZAVODNIK   CAS"
160 FOR i=1 TO z
170 PRINT TAB 3;i;TAB 12;l(i);TAB 20;a(i)
180 NEXT i
```

Pořadí 2.

Po skončení závodu se v libovolném pořadí zadávají dosažené výkony jednotlivých závodníků. Po vložení posledního údaje se zadá "0" a počítač vytiskne výsledkovou listinu. Pokud je lépe hodnocen větší počet dosažených bodů (střelba a pod.), změní se řádek

```
100 IF a(k+1)<a(k) THEN GO TO 150
```

PROGRAM 79

```
 5 REM "PORADI 2"
10 DIM a(50): DIM l(50): LET z=1
20 PRINT "ZAVODNIK","CAS"
30 INPUT "Zavodnik cislo ";l(z)
40 IF L(z)<>0 THEN GO TO 60
50 LET z=z-1: GO TO 90
60 PRINT l(z),
70 INPUT "Jeho cas: ";a(z)
80 PRINT a(z): LET z=z+1: GO TO 30
90 LET c=0: FOR k=1 TO z-1
100 IF a(k+1)>=a(k) THEN GO TO 150
110 LET p=a(k): LET a(k)=a(k+1)
120 LET a(k+1)=p: LET r=l(k)
130 LET l(k)=l(k+1): LET l(k+1)=r
140 LET c=c+1
150 NEXT k
160 IF c<>0 THEN GO TO 90
170 CLS: PRINT "PORADI        ZAVONIK        CAS"
180 FOR i=1 TO z
190 PRINT TAB 3;i; TAB 12;l(i); TAB 20;a(i)
200 NEXT i
```

SPORTKA.

Tento program pomůže váhajícím, jaká čísla mají vsadit do sportky - místo tahání papírků z čepice (se stejným výsledkem).

PROGRAM 80

```
 10 REM SPORTKA
 20 PRINT AT 9,11;",","; AT 10,10;" TYDEN ?"
 30 INPUT t: RANDOMIZE t*100
 40 CLS: DIM a(6)
 50 PRINT AT 4,3;"VYHRAVAJICI CISLA SPORTKY"
 60 PRINT AT 6,6;"PRO ";t;" SAZKOVY TYDEN"
 70 PRINT AT 19,10;"(BEZ ZARUKY)"
 80 FOR n=1 TO 6
 90 LET a(n)=INT (RND *47)+1
100 NEXT n
110 FOR n=1 TO 5
120 LET b=a(1): LET a(1)=a(2): LET a(2)=a(3)
130 LET a(3)=a(4): LET a(4)=a(5): LET a(5)=a(6)
140 LET a(6)=b
150 FOR i=1 TO 5
160 IF a(i)=b THEN LET a(i)=a(i)+1
170 IF a(i)>47 THEN LET a(i)=a(i)-2
180 NEXT i: NEXT n
190 FOR n=1 TO 6
200 PRINT AT 12,1+4*n;a(n);",";
210 NEXT n
220 PRINT AT 21,0;"STISKNI: Z - ZNOVA, K - KONEC"
230 IF INKEY$="z" THEN RUN
240 IF INKEY$<>"k" THEN GO TO 230
250 CLS: PRINT AT 10,7;"SUPERCOMPUTER"
260 PRINT AT 12,8;"SE S VAMI LOUCI"; AT 14,7;"A
    DEKUJE ZA DUVERU."
270 GO TO 270
```

V tomto programu je neobvyklé použití příkazu skoku na řádku 270, tento trik je použit pouze z toho důvodu, aby se jednoduchým způsobem zabránilo výpisu hlášení ukončení programu. Ze smyčky se dostaneme pomocí BREAK, případně můžeme řádek 270 vynechat, nebo zde použít PAUSE 0.

Řádky 20 - 30 upraví tvorbu náhodných čísel tak, aby pro daný týden vycházela vždy stejná. Řádky 50 - 70 vypíší titulek, vlastní tvorba čísel probíhá na řádcích 80 - 100. Dále program zkontroluje, zda se některé číslo nevyskytuje vícekrát stejné - v kladném případě jej upraví (ř. 110 - 180). Tažená čísla vytisknou řádky 190 až 210.

Generátor slov.

Program generuje 2 sloupce pětipísmenných slov. V prvém sloupci jsou uspořádány samohláska - souhláska - samohláska - souhláska, ve druhém sloupci obráceně. Slova se dají použít např. jako názvy výrobků, kodové heslo atd.

PROGRAM 81

```
10 LET a$="AEIOUY"
20 LET b$="BCDFGHJKLMNPQRSTVWXZ"
30 GO SUB 50: GO SUB 50: GO SUB 70: GO SUB 80:
   GO SUB 50: GO SUB 50
40 PRINT : GO TO 30
50 LET a=RND *5+1: LET b=RND *19+1
60 PRINT b$(b)+a$(a);: RETURN
70 LET b=RND *19+1: PRINT b$(b),: RETURN
80 LET a=RND *5+1: PRINT a$(a);: RETURN
```

Program tvoří nekonečnou smyčku, vystoupit z něj se tedy dá pouze příkazem BREAK. Pokud bychom chtěli tvořená slova poněkud "polidštit", můžeme z řetězce b$ vypustit nežádoucí písmena,

jako např. W, Q, X, atd. Naopak žádoucí písmena můžeme vložit vícekrát, aby byly použity častěji. Neopomeneme ale podle délky použitých řetězců upravit konstanty za RND v řádcích 50, 60, a 80.

VERIFIKÁTOR.

Jak je jistě všem uživatelům mikropočítačů známo, je to v podstatě velice rychlý a přesný blbec - provádí pouze to, co do něj programátor vloží, co mu přikáže. Problémy pak nastávají při opisování programů publikovaných v časopisech a při jejich zavádění pomocí klávesnice do počítače, jak jsme se již s tím setkali v kapitole o strojovém kodu. Pro usnadnění této (v podstatě zdlouhavé a málo příjemné) činnosti existují další programy, které vkládaný program kontrolují - vždyt počítač je natolik všestranný nástroj, že by to měl umět, je přece určen k tomu, aby nám ulehčil práci.

Autorem jednoho z těchto programů je Charles Brannon, redaktor časopisu COMPUTE, specialista na problémy software. Původně se mi tento program dostal do ruky v provedení pro počítače ATARI s názvem PROOFREADER, pro který nemáme v češtině dost krátký a výstižný překlad - musíme si tedy bud pomáhat několikaslovným vyjádřením "prověrka vkládaných programů", nebo použít "zkomoleninu" VERIFIKÁTOR, což by mělo značit verifikaci programů.

Program používá speciální kod, něco na způsob kontrolního součtu, s kterým jsme se již setkali při vkládání strojových programů pomocí souboru DATA, zde však funguje i pro programy ve vyšších jazycích (např. BASIC).

Vložíme-li nyní do počítače program, jehož výpis je opatřen kontrolním kodem (tento nevkládáme), můžeme se snadno přesvědčit zda jsou všechny řádky programu zapsány správně.

Použití programu.

Program vkládáme do počítače pomocí již známého loaderu, t.j. programu č.13.

Program vložený do počítače nahrajeme na kazetu a před použitím spustíme příkazem RUN, čímž dojde k uložení strojového kodu do paměti počítače. Následuje přepis původních řádků BASIC verifikátoru vkládaným programem, nebo můžeme řádky vymazat známým způsobem - vložením prázdných řádků (t.j. pouze číslo řádku bez příkazu).

Po vložení celého programu do počítače (bez poznámky REM na konci jednotlivých řádků!!!) spustíme kontrolní program příkazem RANDOMIZE USR 23380 a po příkazu LIST můžeme porovnat kody za poznámkou REM na konci řádků s kody ve výpisu. Pokud kod není shodný, určitě jsme v daném řádku udělali chybu - stačí nyní porovnat pouze vadný řádek s odpovídajícím řádkem ve výpisu.

Po vytisknutí programu na tiskárně zůstane kontrolní kod ve výpisu a tím je usnadněno jeho opětné ukládání. Samotnému programu kontrolní kod nijak nepřekáží - co je uvedeno za výrazem REM, počítač jednoduše ignoruje.

Výhody tohoto kontrolního programu lze nejlépe využít tehdy, jsou-li výpisy programů určené k publikaci opatřeny kontrolním kodem. Pak je úspěšnost aplikace programů zaručena téměř se stoprocentní jistotou.

V této publikaci jsem kontrolní kod u zveřejněných programů nepoužil záměrně, aby jste se mohli pocvičit v pozorném zavádění programů dle předlohy - nakonec, nejsou tak dlouhé, aby je nešlo překontrolovat celé. Každým dalším jejich čtením se vám lépe vštípí do paměti syntaxe jazyka BASIC.

Znovu ale zdůraznuji, že kontrolní kod ani výraz REM do počítače nevkládáme, jelikož by kontrolní program započítal do součtu i kod a výsledek by byl odlišný od uvedeného.

U počítačů ATARI je tento kontrolní kod uveden před číslem řádku, což je výhodnější, počítač si jej pak může přečíst a porovnat se svým. U počítačů řady SINCLAIR a odvozených typů však musí programový řádek vždy začínat číslem, jinak program hlásí chybu.

PROGRAM 82

```
 9 DATA 23296,9
10 DATA "0023E60FC630FE3A3802C60777C98D"
11 DATA "2A4B5CE52A535C18042323232323237D"
12 DATA "A7D1ED521930E9D55E23562B2B2B16"
13 DATA "14AF4F814F7E231D20F91520F67E62"
14 DATA "FEEA20DA790F0F0F0FCD015B79CD015B62"
15 DATA "18CE3AEA00000D00000000DD2100002A3F"
16 DATA "4B5C2BEB2A535CD5CDB819E1A7ED7E"
17 DATA "5219EBDD2330F2DDE5E12422B05C6D"
18 DATA "25110400CDA930444D2A535CCD5516DD5F"
19 DATA "21B05C1323EB23234E2346030303037702BF2"
20 DATA "712B2BEDB0E501050001B214B5BEDB0E1AF"
21 DATA "DD350020E0DD350120DBC30E5B4C",""
```

TELEGRAFNÍ ABECEDA.

Nakonec jsme si nechali jeden delší program, se kterým si dost užijeme. Tento totiž slouží k učení a procvičování příjmu telegrafní abecedy (nesprávně nazývané "morzeovka") a to bud jednotlivých písmen nebo číslic, či celých skupin znaků, případně i delších textů.

Komu by se zdála hlasitost vnitřního reproduktoru v počítači nedostatečná, může si do zdířky EAR nebo MIC (nutno vybrat individuálně) připojit nízkofrekvenční zesilovač, případně vstup GRAMO libovolného radiopřijímače.

Podrobnosti o programu je zde zbytečné vypisovat, neboť jednak jste se již sami dost naučili, aby jste pochopili jak pracuje, jednak vám program sám napoví, co máte dělat.

Tak tedy mnoho štěstí!

PROGRAM 80

```
 1 REM * TELEGRAFIE *
 2 LET a$="NEBYLO NIC ULOZENO": LET b$=a$: LET
   c$=a$
 3 POKE 23658,8: GO TO 100
31 RETURN
32 LET x$="10001"
42 RETURN
43 LET x$="01010"
46 RETURN
47 LET x$="10010": RETURN
48 LET x$="11111": RETURN
49 LET x$="01111": RETURN
50 LET x$="00111": RETURN
51 LET x$="00011": RETURN
52 LET x$="00001": RETURN
53 LET x$="00000": RETURN
54 LET x$="10000": RETURN
55 LET x$="11000": RETURN
56 LET x$="11100": RETURN
57 LET x$="11110"
62 RETURN
63 LET x$="001100"
64 RETURN
65 LET x$="01": RETURN
66 LET x$="1000": RETURN
67 LET x$="1010": RETURN
68 LET x$="100": RETURN
69 LET x$="0": RETURN
70 LET x$="0010": RETURN
71 LET x$="110": RETURN
72 LET x$="0000": RETURN
73 LET x$="00": RETURN
74 LET x$="0111": RETURN
```

```
 75 LET x$="101": RETURN
 76 LET x$="0100": RETURN
 77 LET x$="11": RETURN
 78 LET x$="10": RETURN
 79 LET x$="111": RETURN
 80 LET x$="0110": RETURN
 81 LET x$="1101": RETURN
 82 LET x$="010": RETURN
 83 LET x$="000": RETURN
 84 LET x$="1": RETURN
 85 LET x$="001": RETURN
 86 LET x$="0001": RETURN
 87 LET x$="011": RETURN
 88 LET x$="1001": RETURN
 89 LET x$="1011": RETURN
 90 LET x$="1100"
 99 RETURN
100 CLS: PRINT "UCENI TELEGRAFNICH ZNACEK"
105 PRINT "------------------------------"
110 PRINT AT 3,7;"Recte si vybrat:"
115 PRINT AT 5,2;"1 - Vysilani z klavesnice"
120 PRINT AT 7,2;"2 - Jednotlive znaky"
125 PRINT AT 9,3;"3 - Skupiny znaku"
130 PRINT AT 11,2;"4 - Ulozeni textu A"
135 PRINT AT 13,2;"5 - Ulozeni textu B"
140 PRINT AT 15,2;"6 - Ulozeni textu C"
145 PRINT AT 17,2;"7 - Vysilani textu A"
150 PRINT AT 19,2;"8 - Vysilani textu B"
155 PRINT AT 21,2;"9 - Vysilani textu C"
160 INPUT "Vase volba? ";z
165 LET z=(z+1)*100
170 GO TO z
200 REM - Vysilani z klavesnice -
210 GO SUB 2200: PRINT "NAVRAT ZPET = ENTER"
220 LET a=CODE INKEY$: IF a=13 THEN GO TO 100
```

```
230 IF a>91 OR a<30 THEN GO TO 220
240 GO SUB a: GO SUB 2000
250 PRINT CHR$ a;: GO TO 220
300 REM - Jednotlive znaky -
310 GO SUB 2100: GO SUB 2200
320 PRINT "Nyni budu postupne vysilat"''"25 zneku
    . Kazdy znak kiery"''"uslysis, vloz klavesnic
    i do"''"pocitace.": PAUSE 66: PRINT ''"Pripra
    ven? Stiskni tedy neco."
330 PAUSE 0: FOR Q=1 TO 25
340 LET a=INT (RND*rh)+rl: IF a>rh OR a<rl THEN
    GO TO 440
350 GO SUB a: GO SUB 2000: CLS : PRINT AT 11,8;"
    CO TO BYLO?": PAUSE 0
360 IF CODE INKEY$=a THEN PRINT AT 15,13;"*SPRAVN
    E*": NEXT Q: GO TO 100
370 PRINT AT 15,13;"> CHYBA! <"; AT 18,12;"TO BYL
    O ";CHR$ a: PAUSE 66
380 PRINT AT 21,10;"POSLECHNI SI ZNOVA: ": PAUSE
    66: GO TO 350
400 REM - Skupiny znaku -
410 GO SUB 2100: GO SUB 2200: LET y$=""
420 PRINT "Nyni budu vysilat 25 znaku"''"po petim
    istnych skupinach"''"Vezmi si tuzku a papir p
    ro zapis."; AT 21,0;"Jsi-li pripraven, stiskn
    i tlacitko."
430 PAUSE 0: CLS: FOR Q=1 TO 29
440 LET a=INT (RND*rh)+rl: IF a>rh OR a<rl THEN
    GO TO 440
450 IF Q=6 OR Q=12 OR Q=18 OR Q=24 THEN FOR o=1
    TO 400*t-10: NEXT o: LET y$=y$+"  "
460 GO SUB a: GO SUB 2000: LET y$=y$+CHR$ a
470 NEXT Q: PAUSE 111
480 PRINT "Zkontroluj si zapis."''y$; AT 21,0;"K
    ONEC - stiskni neklere tlacitko."
```

```
 490 PAUSE 0: GO TO 100
 500 REM - Ulozeni textu A -
 510 CLS: INPUT "Vkladej text A ";a$: GO TO 100
 600 REM - Ulozeni textu B -
 610 CLS: INPUT "Vkladej text B ";b$: GO TO 100
 700 REM - Ulozeni textu C -
 710 CLS: INPUT "Vkladej text C ";c$: GO TO 100
 800 REM - Vysilani textu A -
 810 LET z$=a$: GO SUB 2200: GO TO 1100
 900 REM - Vysilani textu B -
 910 LET z$=b$: GO SUB 2200: GO TO 1100
1000 REM - Vysilani textu C -
1010 LET z$=c$: GO SUB 2200
1100 REM - Vysilani textu -
1110 PRINT z$
1120 FOR g=1 TO LEN z$
1130 LET p$=z$(g): LET a=CODE p$
1140 Let n=INT (g/30): LET h=g-n*30
1150 PRINT AT 11,h; CHR$ a; AT 11,h+1;"*"
1160 GO SUB a: GO SUB 2000: NEXT g
1170 PRINT AT 21,0;"KONEC - STISKNI NEKTERY ZNAK"
1180 PAUSE 0: GO TO 100
2000 REM - BEEP znaku -
2010 FOR c=1 TO LEN x$
2020 IF x$(c)="0" THEN BEEP t,10: GO TO 2040
2030 BEEP 3*t,10
2040 FOR o=1 TO 150*t-10: NEXT o: NEXT c
2050 FOR o=1 TO 400*t-10: NEXT o: RETURN
2100 REM - Vyber druhu znaku -
2110 CLS: PRINT AT 6,4;"VYBERTE SI:"
2120 PRINT AT 8,16;"P - Pismena",,"C - Cisla",,"S
     - Smisene"
2130 PAUSE 0: IF INKEY$="P" THEN LET rh=90: LET rl
     =64: RETURN
```

```
2140 IF INKEY$="C" THEN LET rh=57: LET rl=47: RET
     URN
2150 IF INKEY$="S" THEN LET rh=90: LET rl=57: RET
     URN
2160 GO TO 2130
2200 REM - Rychlost-
2210 INPUT "Rychlost - znaku za minutu: ";r
2220 LET t=5/r: RETURN
```

OBSAH

Můj přítel DIDAKTIK GAMA - druhé, upravené vydání.

Dále vyšlo: 100+1 PROGRAMŮ, KLADIVO NA PROGRAMY, MANUÁLY, atd.
K této příručce je možno zakoupit nahrávku VŠECH programů!