5-6/1989

KLUB MIKROELEKTRONIKY

SVAZARM

ATARI®

ZPRAVODAJ

OLOMOUC

Sparta DOS X

Firma ICD (USA) začala prodávat nový výrobek - Sparta DOS X. Je to nový diskový operační systém pro 8-bitové počítače Atari XL/XE. Je založen na cartridgi, což je velice výhodné zejména pro majitele typu 800 XL. Odpadá tím stálé nahrávání DUP systému a systém nabízí nejvíce volné paměti RAM ze všech diskových operačních systémů. Další jeho předností jsou rychlé I/O rutiny. Odborný časopis ANALOG Computing testoval nový výrobek na výstavě fy Atari v Glendale (USA) a byl velmi spokojen s dosaženými výsledky.

Sparta DOS X (dále SDX) je plně kompatibilní s disketovou jednotkou Indus GT, Atari XF 551, US Doubler 1050 a Happy 1050 (poslední dvě jsou upravené disketové jednotky Atari 1050). Všechny záznamové hustoty jsou zde k dispozici (jednoduchá, rozšířená nebo dvojitá). Tedy uživatelé nových disketových jednotek XF 551 mohou konečně využít všech výhod modu DS/DD.

SDX přidává také kompatibilitu s hard-diskem pomocí přídavného zařízení firmy ICD Multi I/O (MIO board). Obsah disku (direktory) může mít až 1423 souborů (ostatní DOSy jsou daleko víc limitovány). Také na cartridgi můžete najít zabudovaný program Archív. Archív (ARC) je program, který kompresuje programy a data do menších souborů pro lepší a rychlejší ukládání na záznamové médium.

Mezi další vynikající vlastnosti lze uvést i podporu až 16 I/O kanálů, až devět disketových jednotek (D9: jako RAMdisk), přímý přístup do RAMdisku kapacity až 1 MB a plnou podporu pro tzv. batch soubory.

Cena cartridge je i s dokumentací $71,95 US. Pro zájemce uvádíme adresu firmy:

ICD
1220 Rock St.
Rockford, IL 61101-1437
USA

Poznámka:

Speciální cartridge SDX stačí zasunout do otvoru pro cartridge na počítačích řady XL/XE. Vysoký komfort obsluhy dokumentuje skutečnost, že SDX má vyvedený konektor pro připojení další cartridge, takže je možno použít i jinou cartridge spolu se systémem SDX.

ANALOG Computing 6/1989
(c) 1989 by GIA Software
Jiří Hrdlička

Kyan PASCAL

Od KYAN PASCALu k TURBO PASCALu!

ing. Jaroslav Choma

Část 3.

Třetí zastavení s jazykem PASCAL začneme problémem, který je stejně starý jako první programátorské pokusy na prvních počítačích, tj. věčný nedostatek volné kapacity paměti.

Každý, kdo si sedne k počítači s rozhodnutím napsat vlastní program, je na začátku okouzlen kapacitou paměti 64 Kb. První pokusy obyčejně dopadnou dobře a náš nadějný programátor začne řešit problém, který jej do této doby tížil. Je přesvědčen, že ATARI je nejlepší počítač na světě a že s jeho pomocí všechno zvládne. Bez velkých úvah začne psát program (v BASICu) přímo do počítače a věří ve své schopnosti. Obyčejně přichází šok hned po napsání prvních řádků, kde dimenzuje pole. Po spuštění této části počítač vypíše našemu programátorovi chybu - nedostatok paměti. Je to pořádná sprcha a ne každý se přes ni lehce přenese. Mnoho začínajících programátorů tato skutečnost odradí od dalších pokusů. Jak se tedy máme zachovat?

Zkušenější vědí o zhušťování údajů - anglicky "pack". V krátkosti bychom si měli vysvětlit pojem zhušťování údajů. Ve standartním BASICu zabírá každá číselná proměnná 6 byte. Je úplně jedno, jeslti je v ní číslo 1 nebo 9.2315E12. V jazyku PASCAL je to řešeno trochu jinak - máme standartní typ dat integer, který zabírá jen 2 byte, a typ real, který podle typu překladače potřebuje od 4 do 8 byte. Co však dělat v případě, kdy jde o reálná čísla? Reálné číslo v Kyan PASCALu totiž zabírá až 8 byte a čísla, která bychom měli zpracovat, jsou sice reálná, ale mají malou hodnotu. Na pomoc si můžeme přibrat modulární koncepci jazyka PASCAL - možnost vytváření procedur a funkcí a možnost definovat vlastní typy dat. Zhušťování dat potom spočívá v tom, že podle velikosti čísel definujeme vlastní typ dat a naše reálná čísla stlačíme procedurou Pack(..). Takto stlačená čísla uložíme. Před zpracováním potom obnovíme hodnotu čísla pomocí procedury UnPack(..). Nutnost použití takové procedury se může projevit nejen při zpracování údajů v paměti, ale i při jejich ukládání na disketu, kde hlavně u vlastních databází můžeme velmi rychle překročit

povolenou kapacitu diskety 127 Kb. Tento problém určitě už přitlačil ty, kteří využívají 8-bitové ATARI pro svou práci.

Problém č.8

Vytvořme procedury Pack a UnPack, které by nám umožnily zpracování čísel v rozmezí -32767.9999 až 32767.9999. Tento rozsah totiž vyhovuje pro zpracování velkého množství problémů, např. počítání finančních prostředků, vyhodnocení měření, databáze různých číselných údajů, atd.

Po krátkém rozboru bychom rychle přišli na to, že časově nejméně náročné řešení je použití typu record, který se skládá ze dvou částí integer. V první části bychom mohli uchovávat číslo před tečkou a v druhé za tečkou. Celkově takto zakódované reálné číslo bude zabírat 4 byte. Vhodné by bylo v případě překročení číselného rozsahu indikovat přetečení pomocí proměnné Err. Procedury nazveme Pack4(..) a UnPack4(..) a mohly by vypadat následovně:

```
procedure  Pack4(Rcislo:real;var  vyst:pack4rc;var
Err:boolean);

const max=3.2768E4;

var R:real;
    a,b:integer;

begin
if ((Rcislo>-max) and (Rcislo<max)) then
 begin
 R:=abs(Rcislo);
 b:=trunc(1E4*(R-trunc(R)));
 if Rcislo>=0 then
  a:=trunc(R)
 else
  begin
  if trunc(R)=0 then
   begin
   a:=0;
   b:=-b
   end
  else
   a:=-trunc(R)
  end;
 Err:=false;
 with vyst do
  begin
  predb:=a;
  zabod:=b
  end
 end
else
 Err:=true
end;
```

```
procedure UnPack4(vst:pack4rc;var Rcislo:real);

var a:integer;

begin
if vst.predb<0 then
 Rcislo:=vst.predb-vst.zabod/1E4
else
 Rcislo:=vst.predb+vst.zabod/1E4
end;
```

Před použitím těchto procedur musíme mít definovaný type pack4rc jako record, který se skládá ze dvou částí - predb a zabod typu integer. Tyto procedury nám pro uvedený rozsah reálných čísel umožní zpracovat až dvakrát více čísel. Na krátkém programu si můžeme ukázat použití takto vytvořených procedur:

```
program spck4;

type pack4rc=record
 predb:integer;
 zabod:integer
 end;

var r:real;
    e:boolean;
    p:pack4rc;

#I D8:PCKUNPC4.I

begin
write( zadej cislo  );
readln(r);
Pack4(r,p,e);
if not e then
 begin
 r:=0;
 UnPack4(p,r);
 writeln( po pack a unpack.. ,r:10:5)
 end
else
 writeln( byla chyba..r )
end.
```

Princip programu je jednoduchý - po zadání reálného čísla z určeného intervalu převedeme zhušťování dat na 4 byte a pokud nebylo číslo mimo rozsah, dekódujeme ho zpět do real. V případě, že číslo bylo mimo rozsah, vypíše program nápis "byla chyba..r". Program je třeba chápat jen jako důkaz, že procedury pracují správně. Ve skutečném použití nemusí být obě procedury v jednom programu. Jako příklad si můžeme uvést databázový program, kde jeden program slouží na

zapisování údajů do databáze na disketu a druhý tyto údaje zpracovává.

Často však potřebujeme zpracovávat čísla z ještě menšího rozmezí, např. -127.99 až 127.99...

Problém č.9

Vytvořme procedury pro zpracování reálných čísel z intervalu -127.99 až 127.99. Vzhledem k rozsahu čísel bude stačit nový typ pack2rc definovaný pomocí standartního typu integer. Podobně jako u předchádzejících procedur budeme uchovávat zvlášť číslo před tečkou a zvlášť číslo za tečkou. Procedury by mohly vypadat takto:

```
procedure  Pack2(Rcislo:real;var  vyst:pack2rc;var
Err:boolean);

const max=128;

var R:real;
  a,b:char;
    c:integer;

begin
if ((Rcislo>-max) and (Rcislo<max)) then
 begin
 R:=abs(Rcislo);
 b:=chr(trunc(100*(R-trunc(R))));
 if Rcislo>=0 then
  a:=chr(trunc(R))
 else
  a:=chr(max+trunc(R));
 Err:=false;

#a
 LDY #$6
 LDA (SP),Y
 LDY #$3
 STA (SP),Y
 LDY #$5
 LDA (SP),Y
 DEY
 STA (SP),Y
#

 vyst:=c
 end
else
 Err:=true
end;

procedure UnPack2(vst:pack2rc;var Rcislo:real);

var c,a,b:integer;
```

```
begin
c:=vst;

#a
 LDY #$7
 LDA (SP),Y
 LDY #$5
 STA (SP),Y
 INY
 LDA #$0
 STA (SP),Y
 LDY #$8
 LDA (SP),Y
 LDY #$3
 STA (SP),Y
 INY
 LDA #$0
 STA (SP),Y
#

if a>=128 then
 Rcislo:=-((a-128)+b/100)
else
 Rcislo:=a+b/100
end;
```

V těchto procedurách je použit assembler hlavně pro zefektívnění práce procedur. Vlastní logika je napsaná v PASCALu. V assembleru jsou jen 2 části, jedna rozděluje údaje do jednotlivých byte a druhá vybírá jednotlivé byte v proceduře UnPack. Samozřejmě je možné tyto procedury vytvořit jen v PASCALu. Pomocí příkazů jazyka je výhodné tvořit procedury a funkce tehdy, když uvažujeme o přenositelnosti na jiné typy počítačů s překladači jazyka PASCAL. Je ale jasné, že použitím assembleru zvyšujeme efektívnost programu. Při použití assembleru však ztrácíme jednu z nejdůležitějších výhod - přenositelnost programů. Při přenosu programů to řešíme vytvořením procedury a funkce, které realizují téže funkce, ale s použitím prostředků pro daný počítač a překladač. Výjimkou jsou například procedury a funkce jazyka Turbo PASCAL, které vytváříme ne proto, abychom je přenášeli na jiné počítače, ale proto, abychom pro Kyan PASCAL vytvořili takové prostředí, které by nám umožnilo vytvářet programy využitelné v Turbo PASCALu.

Vraťme se však k našemu tématu - Pack a UnPack pro čísla z rozsahu -127.99 až 127.99. Použití procedur si opět ukážeme na jednoduchém programu:

```
program spck2;

type pack2rc=integer;
```

```
var r:real;
    e:boolean;
    p:pack2rc;

#I D8:PCKUNPC2.I

begin
write( zadaj cislo  );
readln(r);
Pack2(r,p,e);
if not e then
 begin
 r:=0;
 UnPack2(p,r);
 writeln( po pack a unpack.. ,r:10:5)
 end
else
 writeln( byla chyba..r )
end.
```

O programu není nutné mluvit, protože je jen variantou programu spck4. Mezi obdobné problémy můžeme zařadit ukládání a zpracování celočíselných údajů, kdy rozsah těchto čísel nepřekročí -127 až 127.

Problém č.10

Realizujme procedury pro zhušťování celočíselných údajů z daného intervalu -127 až 127. Vzhledem k rozsahu čísel využijeme nový typ pack1i definovaný pomocí standartního typu char:

```
procedure  Pack1(cvst:integer;var  vyst:pack1i;var
Err:boolean);

const max=128;

begin
if ((cvst>-max) and (cvst<max)) then
 begin
 if cvst>=0 then
  vyst:=chr(cvst)
 else
  vyst:=chr(max+abs(cvst));
 Err:=false
 end
else
 Err:=true
end;

procedure UnPack1(vst:pack1i;var icislo:integer)


const max=128;
```

```
var pom:integer;

begin
pom:=ord(vst);
if pom>max then
 icislo:=max-pom
else
 icislo:=pom
end;
```

Použití procedury nám ušetří 1 byte pro každý celočíselný údaj, který uložíme pomocí této procedury. Použití si můžeme ukázat na příkladu:

```
program spck1;

type pack1i=char;

var i:integer;
    e:boolean;
    p:pack1i;

#I D8:PCKUNPC1.I

begin
write( zadej cislo  );
readln(i);
Pack1(i,p,e);
if not e then
 begin
 i:=0;
 UnPack1(p,i);
 writeln( po pack a unpack.. ,i:10)
 end
else
 writeln( byla chyba..i )
end.
```

Program využívá stejný mechanismus jako v předcházejícím případu. Důležité je nezapomenout definovat pro každý program, který využívá některé z procedur, nové typy pack4rc, pack2rc nebo pack1i!

Zatím jsme mluvili hlavně o té lepší stránce použití těchto procedur - o úspoře paměti. Otázkou je, jakou zaplatíme daň za jejich použití. Samozřejmě výrazné bude hlavně zpomalení vykonávání programu. Nejkritičtější bude pravděpodobně u procedur Pack4 a UnPack4, protože jsou nejnáročnější na výpočty. Prodloužení zpracování po použití procedur Pack a Unpack si můžeme ukázat na programu pomocí procedur pro čas TIME.I:

```
program sr;
```

```
var r:real;
    i,min,sek:integer;

#I D8:TIME.I

begin
SetTime;
for i:=1 to 1000 do
 r:=i;
Time(min,sek);
writeln( min.. ,min,  sek.. ,sek)
end.
```

Program sr ukazuje použití bez zhušťování údajů. Vidíme, že výsledný čas je do 2 sekund. Ukažme si obdobu s použitím zhušťování dat:

```
program scasupck;

type pack4rc=record
     predb:integer;
     zabod:integer
     end;

var min,sek,i:integer;
    r:real;
    e:boolean;
    pc:pack4rc;

#I D8:PCKUNPC4.I
#I D8:TIME.I

begin
SetTime;
for i:=1 to 1000 do
 begin
 r:=i;
 Pack4(r,pc,e);
 UnPack4(pc,r);
 end;
Time(min,sek);
writeln( min.. ,min,  sek.. ,sek)
end.
```

Po spuštění tohoto zkušebního programu vidíme, že se prodloužil čas zpracování až na 22 sekund. Z toho je jasné, že použití těchto procedur je opodstatněné jen ve skutečně nevyhnutelných případech. Na podobných principech můžeme vytvořit i další procedury, např. na zhuštění logických hodnot. Vždyť na uchování této informace potom postačí jen 1 bit a do 1 byte potom můžeme uschovat až 8 logických hodnot. Zde se může uplatnit právě assembler, protože přečtení bitu nebude v assembleru dělat velké problémy.

Část 4.

Setkáváme se už počtvrté při jazyku Kyan PASCAL a věřím, že se pro mnohé stal jazykem, kterému se budou dále věnovat. Opět je třeba zdůraznit hlavně modulárnost jazyka PASCAL jako jednu z nejvýznamnějších vlastností tohoto jazyka jako i všech dalších jazyků, které si z PASCALu berou vzor. V předcházejících částích jsme se věnovali různým užitečným procedurám a funkcím, které ale nevyužívaly práci se soubory. V této části se dotkneme i této problematiky. Nebude to však lekce o používání souborů, ale jen popis některých zajímavých možností práce se soubory pro verzi překladače Kyan PASCAL. V případě zájmu těch, kteří ještě se soubory nepracovali, mohu doporučit učebnici Jinoch, Muller, Vogel: Programování v jazyku PASCAL.

Problém č.11

Standartně jazyk PASCAL pracuje se dvěma soubory - INPUT a OUTPUT. Tyto jsou (pokud nedefinujeme jinak) přiřazeny editoru E:. Jak ale například poslat text nebo výsledky výpočtů na tiskárnu? Na disketě má každý procedury pod názvem PR.I nebo PRINTER.I. Jsou to dvě procedury, kde jedna má za úlohu spustit tisk údajů, které standartně směřují na E:, a druhá má tisk údajů zastavit. Procedury na disketě jsou napsané v assembleru a jejich použití značně ztěžuje přenositelnost programů na jiné počítače s jiným překladačem. I když v manuálu jazyka Kyan PASCAL (vlastním týden návod přeložený v AK Hodonín) není jiný způsob uveden, přesto existuje!

Pro tiskárnu můžeme použít standartní proceduru rewrite. Na jednoduchém příkladu si můžeme ukázat použití procedury při tisku hodnot:

```
program spr;
var pok,inp:file of char;
    j:integer;
    r:real;
begin
rewrite(pok, P: );
J:=0;
repeat
 j:=j+1;
 r:=1/j;
 writeln(pok, j= ,j:3,  r= ,r:10:5)
until j=20
end.
```

Proč jsou ale potom na disketě procedury pro práci s tiskárnou? Pravděpodobně proto, že při použití těchto procedur můžeme sledovat, co právě tiskneme, a také proto, že nemáme stále obsazený kanál, který ještě můžeme pro další práci potřebovat. PASCAL (alespoň ten standartní) má totiž jednu nepříjemnou vlastnost - hned po použití reset nebo

rewrite máme takto nadefinovaný kanál obsazen až do ukončení programu.

V tomto ukázkovém programu jsme použili textový soubor. Měli bychom si tedy objasnit místo textových souborů v jazyku PASCAL. Pověděli jsme si, že PASCAL má nadefinované standartní soubory - INPUT a OUTPUT. V podstatě i tyto soubory jsou textové. K čemu vlastně slouží? INPUT představuje vstup z určitého zařízení, přičemž automaticky dochází ke konverzi údajů z textové formy do vnitřní podle typu proměnných, do kterých údaje snímáme. OUTPUT představuje výstup údajů v textové podobě prostřednictvím automatické konverze z vnitřního tvaru podle typu proměnné a požadavků na výstup - celkový počet míst, počet desetinných míst... Textové soubory jsou významné hlavně proto, že umožňují přenos údajů mezi různými programovacími jazyky a programy. Představme si například situaci, že potřebujeme předat údaje z BASICu do PASCALu. Jak realizovat výměnu parametrů?

V BASICu máme prakticky dvě možnosti zápisu údajů do souborů - jeden pracný pomocí příkazu PUT a znalosti adresy proměnné v paměti a druhý jednoduchý pomocí PRINT #č... V prvním případě narazíme na problém neslučitelnosti dat v BASICu a v PASCALu, protože každá číselná proměnná v BASICu zabírá 6 byte a v PASCALu jsou standartní dva typy: integer a real. Typ integer zabírá 2 byte a real 8 byte. Pravděpodobně by bylo velmi náročné vytvořit proceduru pro převod dat. Reálná je tedy prakticky jen druhá možnost - přenos přes textový soubor. Dříve než si ukážeme načítání dat z textového souboru, ukážeme si zápis do tohoto souboru pomocí krátkého programu:

```
program stxtw;
var pok:file of char;
    j,i:integer;
    r:real;
begin
rewrite(pok, D8:INPUT.TXT );
j:=0;
repeat
 j:=j+1;
 r:=1/j;
 writeln(pok,j:3,r:10:5)
until j=20
end.
```

Ekvivalentní textový soubor můžeme získat pomocí zápisu v BASICu, ale také ze SynFile+, případně SynCalc, SpeedCalc... Zde je třeba říci, že častým problémem v amatérské, ale i v profesionální praxi je zpracovaní a údržba dat pomocí databázových a kalkulačních programů. Tyto programy jsou výkonnými pomocníky, ale často neumožňují některé náročné operace s daty. Jako výhodné se jeví kombinovat editační možnosti databází se schopnostmi kalkulačních programů a pomocných programů v některém z

vyšších jazyků. Pro ATARI bylo vytvořeno více vhodných databázových programů - SynFile+, MiniOffice II a také řada kalkulačních programů - SynCalc, SpeedCalc, VisiCalc. Prakticky všechny programy mají možnost uložit údaje v textové formě. V SynFile+ to bude pomocí LISTS. Po volbě LISTS zadáme, které položky chceme uložit, jak bude dlouhá stránka a titul. Jako výstupní zařízení zvolíme samozřejmě disk. Po zadání kritérií pro vyhledávání budou v textové formě zapsané všechny vyhovující údaje na disketu. Pro názornost si můžeme ukázat výstup těchto údajů:

VÁHA	MNOŽSTVÍ
0.02	453
0.93	9
1.23	45
5.43	98
9.23	123
12.13	234
12.23	234
34.11	568
34.21	932
44.99	453
95.00	451

Před zpracováním programů potom můžeme odstranit hlavičky a řádky, které by vadily při zpracování. Samozřejmě podle toho, jak nadefinujeme pořadí výstupních údajů do textového souboru z databáze, musíme nadefinovat i proměnné nutné pro načtení údajů. Pro soubor vytvořený programem stxtw by potom načítání údajů mohlo vypadat následovně:

```
program stxtr;
var inp:file of char;
    i:integer;
    r:real;
begin
reset(inp, D8:INPUT.TXT );
while not eof(inp) do
 begin
 readln(inp,i,r);
 writeln( i= ,i:3,  r= ,r:10:5)
 end
end.
```

Pro textový soubor z databáze by samozřejmě program nevyhověl, protože na ekvivalentních místech nejsou ekvivalentní typy proměnných. Přepracování programu by však nemělo dělat velké problémy. Obdobné možnosti máme i v případě textového výstupu z kalkulačních programů, např. programu SpeedCalc. Zde vytiskneme údaje na disketu a tyto už bez velkých problémů dokážeme načíst. Výstup dat na disketu má ještě jednu dost podstatnou výhodu - takto

vytvořené údaje můžeme dále zpracovávat pomocí textového editoru. Účelná kombinace možností umožňuje skutečně využít vlastnosti našich počítačů i při přípravě článků pro Zpravodaj a obcházet tak omezení jednotlivých programů. Bez velkých problémů můžeme například počítat různé údaje pomocí programu v PASCALu a vypočítané výsledky zapisovat v textové formě na disketu. Údaje pro zpracování mohou být načtené ze souborů vytvořených pomocí databázového programu SynFile+, SpeedCalcu, BASICu... Velké možnosti se tady nabízejí studující mládeži při psaní diplomových prací, různých soutěžních prací, ale také v profesionální praxi při psaní výzkumných zpráv, atd. Výsledky výpočtů můžeme ukládat buď do souboru na disketě, nebo je můžeme přímo vytisknout na tiskárně. Na příkladě si můžeme ukázat kombinaci načtení a tisku textových souborů. Program vyžaduje tiskárnu:

```
program sprinter;
var pok,inp:file of char;
    i:integer;
    r:real;
begin
rewrite(pok, P: );
reset(inp, D8:INPUT.TXT );
while not eof(inp) do
 begin
 readln(inp,i,r);
 writeln(pok, i= ,i:3,  r= ,r:10:5)
 end
end.
```

Program může sloužit jako ukázka práce se dvěma textovými soubory, kde jeden je vstupní a druhý výstupní. Princip spočívá v tom, že postupně až do konce souboru načteme údaje a tyto "zobrazíme" na tiskárně. Mezi podobné problémy můžeme zařadit i načítání znaku při stisku klávesy.

Problém č.12

Vytvořte pomocí standartních procedur pro práci s textovými soubory možnost snímání kódu po stisku klávesy. Verze jazyku Kyan PASCAL umožňuje realizovat snímání stisknuté klávesy velmi jednoduše pomocí textového souboru nadefinovaného pro standartní zařízení K:, se kterým spolupracuje OS. Podobně je možná spolupráce s kazetovým magnetofonem, případně libovolným zařízením, které je nadefinované v tabulce obsluhy. Snímání kódu klávesy ukazuje tento příklad:

```
program skl;
var kl:file of char;
begin
reset(kl, K: );
repeat
```

```
 writeln( kl.. ,kl );
 get(kl)
until kl = K
end.
```

Program je jednoduchý - až do zadání "K" čeká na stisknutí klávesy a po stisknutí ukáže, která klávesa byla stisknutá.

Bylo by snad dobré povědět si i o nevýhodách zpracování textových souborů. Nejpodstatnější bude pravděpodobně rychlost zpracování, protože při zpracování textových souborů jsme omezeni na sekvenční zpracování. Verze Kyan PASCAL však umožňuje i zpracování souborů s přímým přístupem k libovolné položce. I když jsou údaje zapsané "textově", můžeme vytvořit proceduru Val, která by realizovala převod textově zapsaného čísla do real, případně integer... To by umožnilo podstatně zrychlit zpracování hlavně u souborů vytvořených pomocí databázových programů, kde už jsou údaje utříděné, což může ulehčit zpracování. Příkladem nám může být Turbo PASCAL, kde pro zpracování čísel zapsaných textově existují procedury Str (=pro zápis reálného čísla do stringu) a Val (=pro dekódování čísla ze stringu). V současnosti nejsou vytvořeny ekvivalenty těchto procedur, které by efektivně řešily problém převodu čísla na řetězec a naopak. Po jejich dokončení budou opět uveřejněné prostřednictvím Zpravodaje. (Výzva pro další programátory v jazyku Kyan PASCAL.)

Uvedené nejsou samozřejmě všechny možnosti práce s textovými soubory, ale i to, co je v tomto příspěvku popsané, umožní bez velkých obav používání vymožeností výkonného OS ATARI ve spolupráci s Kyan PASCALem. Samostatnou kapitolu tvoří netextové soubory, ale vzhledem k úplné shodě práce s těmito soubory jako se standartem, není účelné se těmto souborům věnovat. Zájemci o práci s těmito soubory najdou dostatek informací v odborné literatuře.

Část 5.

Pátou část setkání s jazykem Kyan PASCAL začneme neobvykle. Nebudeme chválit, ale právě naopak budeme hovořit o chybách a nedostatcích této verze. Nejzávažnější chybou, která může každého potkat, je špatné ošetření při dělení číslem 0 nebo při dělení proměnnou, ve které je 0 (S touto chybou jsem se setkal při svých prvních pokusech a dlouho jsem nevěděl, co se vlastně stalo.). Chyba se projevuje tak, že po dělení nulou vypadne systém a počítač "ztvrdne". Tato chyba je v knižním modulu LIB, kde je tato situace špatně ošetřená. Před krátkým časem bylo v časopise BAJTEK č.4/1988 uveřejněné ošetření této chyby. Ošetření spočívá v zavedení modulu, který má na starosti přepsání registrů v knižním modulu LIB. Modul je nutné zařadit do textu programu v deklarační části a vypadá následovně:

```
(*modul KorErr*)
```

```
#a
 LDA #0
 STA 39020
 LDA #216
 STA 39021
 LDA #>39021
 STA 40398
 LDA #<39021
 STA 40399
#
```

V tomto časopisu je popsáno také řešení s opravou modulu LIB přímo na disketě. Výhodnější však bude zavedení KorErr, protože pro vlastníky diskety je už verze Kyan PASCAL 2.0 bez této chyby a vlastníkům kazetové verze je řešení s přepsáním na disketu nepřístupné.

S další nepříjemnou chybou se můžeme setkat při práci s procedurami a funkcemi při předávání parametrů. První, co může zarazit i zkušeného programátora, je to, že překladač špatně kontroluje počet předávaných parametrů. Mějme např. definovanou proceduru:

```
procedure I(a,b,c:integer);
    .
begin
 .
 .
end;
```

Po jejím vyvolání v programu s nesprávným počtem parametrů program pokračuje dále, ale samozřejmě s chybnými údaji, např.:

```
begin
 .
I(par1,par2);
 .
end.
```

Pozor na tuto chybu, protože překladač ji neodhalí a chybný počet parametrů se dá lehce přehlédnout! Na tuto chybu, vzhledem k tomu, že je způsobena překladačem, neexistuje ošetření.

Předávání parametrů se týká i další chyba, a to špatná kontrola shodnosti typu při předávání parametrů. Mějme např. proceduru:

```
procedure a(z:string12);
 .
begin
 .
 .
end;
```

String12 by byl nadefinovaný jako: type string12:array 1..12 of char;. Při přiřazení z:=b, kde b by bylo jiného typu, překladač neshodu odhalí, ale v případě vyvolání procedury s parametrem neshodného typu se nic neděje a překlad ukončí činnost obvyklým způsobem. Problémy však nastanou při provozu programu v případě, kdy budeme testovat shodu nebo řětězec (string) bude vstupem pro nějakou proceduru pracující se soubory na disketě. Vyvolání procedury může vypadat takto:

```
begin
 .
a( meno );
 .
 .
end.
```

Tato chyba je také chybou překladače a je prakticky neodstranitelná. Na příkladu si můžeme ukázat, jak se projevuje:

```
program sstring;
type str=array 1..12 of char;

procedure tlac(ret:str);
var i:integer;
begin
for i:=1 to 12 do
 writeln( znak  ,i, ... ,ret i );
end;

begin
tlac( ahoj )
end.
```

Při překladu tohoto programu překladač neodhalí neshodu v délce řetězce a skončí standartně. Po jeho spuštění si můžeme všimnout, že po výpisu posledního platného znaku se objeví různé ATASCII znaky. Tato chyba je poměrně závažná, protože právě kontrola shodnosti typu patří mezi největší výhody PASCALu.

Nemusíme však zoufat, ale musíme se naučit "žít" i s těmito chybami. Je třeba říci i to, že ve verzi 2.0 a vyšších by všechny tyto chyby měly být odstraněny. Problémem budou tyto chyby pro vlastníky kazetových verzí, protože vzhledem k novému způsobu práce ve verzi 2.0 a vyšších (překlad probíhá ve dvou etapách - do makroassembleru a potom do tvaru .OBJ), nebude pravděpodobně přístupná pro vlastníky kazetového magnetofonu a ti budou dále odkázáni na verzi 1.0 až 1.3.

V další části našeho setkání s PASCALem se budeme věnovat vývoji procedur pro práci se soubory - Rename, Delete, Lock... Operační systém našeho ATARI (mimochodem patří mezi nejvýkonnější OS ve třídě 8-bitových počítačů)

nabízí jednoduchou realizaci i ekvivalentu IOresult - procedury pro zjištění vstupně/výstupní chyby. Pro pochopení následujícího programu je třeba si vysvětlit možnosti realizace těchto procedur a také objasnit konkrétní provedení. První, co si musíme říci, je to, že tyto procedury by měly být univerzální a jejich použití by mělo být co nejjednodušší. Na pomoc bychom si měli vybrat práci se soubory v Turbo PASCALu a Turbo BASICu. Hned si však musíme uvědomit, že vyvolání procedury stylem Delete(MENSUB) bude těžko realizovatelné, protože PASCAL kontroluje shodnost typů (alespoň standartní PASCAL zcela určitě a my bychom kvůli slučitelnosti neměli využívat chyby Kyan PASCALu) a je těžké zvolit takový typ pole, který by vyhovoval pro různé délky jména. Neobejdeme se zde také bez použití assembleru. Výhodné by bylo využít to, že po každé vstupně/výstupní operaci OS nastavuje v registrech IOCB použitý kód chyby. Měli bychom ještě rozhodnout, jak zvolíme IOCB, který budeme používat. Přidělování kanálů v PASCALu automaticky dělá kompilátor a my můžeme buď využít této znalosti, anebo využít kanál č.0, protože během práce těchto procedur pravděpodobně obrazovku používat nebudeme. Pro náš případ si zvolíme IOCB č.0. Potřebné je ještě vytvoření podprogramů v assembleru, které by realizovaly některé částečné úlohy. Podprogramy by mohly vypadat takto:

```
#a
ZAC JSR COPY
 LDX #$0
 JSR CL
 JSR MENO
 RTS
;
KON LDA #$0
 STA $34A,X
 STA $34B,X
 JSR $E456
 JSR SET
 JSR OP0
 RTS
;
COPY LDX #$0
 LDY #$7
N4 LDA (SP),Y
 CMP #$20;" "
 BNE N5
 LDA #$9B;RET
 STA MEN,X
 JMP N6
N5 STA MEN,X
 INY
 INX
 CPX #$1C;..28
 BNE N4
N6 NOP
```

```
 RTS
;
SET LDA $343,X
 LDY #$3
 STA (SP),Y
 RTS
;
MENO LDA #>MEN
 STA $344,X
 LDA #<MEN
 STA $345,X
 RTS
;
MEN DW $0
 DW $0
 DW $0
 DW $0
 DW $0
 DW $0
 DW $0
 DW $0
 DW $0
 DW $0
 DW $0
 DW $0
 DW $0
 DW $0
 DB $9B;RET
;
CL0 LDX #$0
 JSR CL
 RTS
;
CL LDA #$C
 STA $342,X
 JSR $E456
 RTS
;
OP0 LDX #$0
 LDA #$3
 STA $342,X
 LDA #>E
 STA $344,X
 LDA #<E
 STA $345,X
 LDA #$C
 STA $34A,X
 LDA #$0
 STA $34B,X
 JSR $E456
 RTS
;
E DB $45;E
 DB $3A;:
```

```
 DB $9B;RET
;
#
```

Měli bychom si objasnit účel některých podprogramů, které jsou zde uvedené. Podprogram COPY má na starosti překopírování názvu souboru do pracovního zásobníku. Nastavení některých parametrů ještě nebudeme vysvětlovat, protože ty se vyjasní až po uvedení procedury Delete(..). Závislé jsou na umístění parametrů procedury nad SP.

COPY kopíruje maximálně 28 znaků. Tato délka závisí na délce názvu souboru v nejhorším případě, kterým je procedura Rename(..). Zde může být až 28 znaků. Jako zásobník jména slouží MEN, na konci kterého je ještě znak $9B představující return.

SET je podprogram nastavující do pomocné proměnné kód chyby po vstupně/výstupní operaci. Kódy odpovídají chybám platným pro BASIC a hodnota 1 naznačuje, že všechno proběhlo bez problémů.

MENO má na starosti nastavení ukazovatele na zásobník, ve kterém je název souboru pro zpracování. Pro pochopení můžeme doporučit nastudování CIO rutiny.

CL0 a CL realizují uzavření kanálu. V CL0 sa nastavuje kanál, který má být uzavřen, a CL uzavření vykoná.

OP0 otevírá IOCB č.0 opět pro editor. Při otevírání nastavuje všechny hodnoty platné pro editor E:. Písmeno E zde představuje jméno editoru E:.

Důležité jsou podprogramy ZAC a KON, které slouží k vyvolání jednotlivých podprogramů. Celá filozofie podprogramů je postavená na použití IOCB č.0. V případě, že nebude tato volba vyhovovat, můžeme vyhledat volný kanál pomocí hodnoty prvého byte IOCB. Nemělo by to dělat velké problémy, ale tato možnost nebyla odzkoušená. Použití podprogramů v assembleru si můžeme ukázat na proceduře Delete(..):

```
procedure Delete(meno:FNAME;var Err:integer);
var c:integer;
begin
c:=0;
#a
 TXA
 PHA
 JSR ZAC
 LDA #$21
 STA $342,X
 JSR KON
 PLA
 TAX
#
Err:=c
end;
```

Využíváme zde pomocné proměnné c pro kód chyby. Po

úschově hodnoty X registru vyvoláme ZAC. Do akumulátoru vložíme hodnotu $21 (desítkově 33), která představuje kód operace delete, zavoláme KON - podprogram realizující vlastní volání CIO, nastavení kódu chyby a opětovné otevření kanálu č.0 pro editor E:. Použití takto nadefinované procedury si můžeme ukázat na programu:

```
program sdel;

type FNAME=array 1..6 of char;

var M:FNAME;
   ch:integer;

(*zde vložit modul ASS.I*)
#I D8:DELETE.I;

begin
writeln( pozor program pracuje s IOCB0 );
readln;
Delete( D:POK ,CH);
writeln( chyba.. ,ch)
end.
```

V programu si můžeme všimnout, že se neshoduje délka názvu souboru s nadefinovaným typem. Program i přesto bude přeložený bez chyb a bude fungovat. Tuto "fintu" však už nemůžeme použít u vyšších verzí. Tam už bude třeba dodržet délku řetězce a zbytek řetězce do nadefinované délky vyplnit mezerami. Na úplně stejném principu můžeme nadefinovat i další procedury realizující některé důležité operace. Podle vzoru Turbo PASCALu můžeme nadefinovat tzv. BOX, který můžeme nazvat například MINIDOS.I. Zde budou shrnuty všechny potřebné procedury pro práci se soubory:

```
(* zde vložíme ASS - modul podprogramů, které jsme
už uvedli *)
(* proceduru Delete jsme už také uvedli *)

procedure Rename(meno:FNAME;var Err:integer);
var c:integer;
begin
c:=0;
#a
 TXA
 PHA
 JSR ZAC
 LDA #$20
 STA $342,X
 JSR KON
 PLA
 TAX
#
Err:=c
```

```
end;

procedure Protect(meno:FNAME;var Err:integer);
var c:integer;
begin
c:=0;
#a
 TXA
 PHA
 JSR ZAC
 LDA #$23
 STA $342,X
 JSR KON
 PLA
 TAX
#
Err:=c
end;

procedure UnProtect(meno:FNAME;var Err:integer);
var c:integer;
begin
c:=0;
#a
 TXA
 PHA
 JSR ZAC
 LDA #$24
 STA $342,X
 JSR KON
 PLA
 TAX
#
Err:=c
end;

procedure SFormat(meno:FNAME;var Err:integer);

var c:integer;
begin
c:=0;
#a
 TXA
 PHA
 JSR ZAC
 LDA #$FD
 STA $342,X
 JSR KON
 PLA
 TAX
#
Err:=c
end;
```

```
procedure DFormat(meno:FNAME;var Err:integer);
var c:integer;
begin
c:=0;
#a
 TXA
 PHA
 JSR ZAC
 LDA #$FE
 STA $342,X
 JSR KON
 PLA
 TAX
#
Err:=c
end;
```

Procedury jsou vlastně shodné až na kód operace vložené prostřednictvím akumulátoru do registru $342, ve kterém se nastavuje X (jaká operace má být vykonána).

Pokračování příště.

MiSe - Atari BASIC Revision B Fix
Matthew J.W. Ratcliff

Pokud máte ve svém počítači Atari XL zabudovánu nepříliš kvalitní verzi B programovacího jazyka Atari BASIC, nemusíte si zoufat. "Kouzelník" Matt opět dokázal nemožné. Krátký program, který najdete ve výpisu 1, "opraví" váš Atari BASIC Revision B překopírováním BASIC ROM do RAM a zapsáním správných bytů do příslušných míst. To znamená, že nyní je BASIC pozměnitelný, protože se nachází v RAM. Tak se doplní BASIC B na BASIC C - bez nutnosti mít příslušný ROM chip nebo cartridge.

Při typování programu postupujte opatrně. Vybírejte si jen ty řádky, které bude potřebovat vaše záznamové médium (disketa nebo kazeta). Patřičné pokyny jsou uvedeny v instrukcích REM.

Po spuštění programu příkazem RUN vygeneruje program rutinu ve strojovém kodu. Uživatelé disketových jednotek obdrží soubor AUTORUN.SYS. Majitelé kazetových magnetofonů nahrají rutinu zpět do počítače tímto způsobem - zapnutím počítače se stisknutým tlačítkem START.

Tento program je vhodné uložit na začátku každé strany diskety nebo kazety, která obsahuje programy v jazyku Atari BASIC.

Překlad některých komentářů REM program MLBTOC.BAS:
Tento program při RUN vytvoří vlastní soubor AUTORUN.SYS.
Doporučuje se posunout dolů RAMTOP, protože některé
grafické příkazy vynulují RAM nad RAMTOPem. (viz příklady)

Mapping the Atari 1985
(c) 1989 by GIA Software
Jiří Hrdlička

Listing 1 - program MLBTOC.BAS Part: 1

```
QY 10 REM ┌──────────────────────────┐
IH 20 REM |ATARI 600/800 XL REV.B(UGS)|
VE 30 REM | BASIC TO REV.C CONVERTER  |
RB 40 REM └──────────────────────────┘
FP 50 REM By Matthew J.W. Ratcliff
PM 60 REM (c) 1985
UI 70 REM This loader will create
GC 80 REM an AUTORUN.SYS for you.
ZU 90 REM For disk drive or datase users.
LH 100 REM Caution:
KT 110 REM Advisable to move down RAMTOP
IO 120 REM when in the RAM/BASIC, since
XT 130 REM some Atari graphics commands
GG 140 REM will clear RAM above RAMTOP.
QP 150 REM i.e. POKE 106,PEEK(106)-4:GRAPHICS 0-6
AY 160 REM i.e. POKE 106,PEEK(106)-16:GRAPHICS 7-11
MH 170 REM Disk drive - lines 200-350 and data line 1010!
JM 180 REM DatasetMGF - lines 400-550 and data line 1010!
YK 200 REM Disk Drive
CD 210 RESTORE
QE 220 GRAPHICS 0:DIM A$(10)
QN 230 ? "GET DOS DISK READY FOR REV.B TO C"
JE 240 ? "AUTORUN FILE AND PRESS RETURN KEY"
BR 250 TRAP 350:INPUT A$
WH 260 OPEN #1,8,0,"D:AUTORUN.SYS"
XA 270 READ A:IF A<0 THEN 290
ZR 280 PUT #1,A:GOTO 270
GD 290 CLOSE #1:? "*** ALL DONE ***"
AE 300 ? :? "SAVE THIS LOADER AS A BACKUP":? "JUST IN CASE!":EN
       D
OP 350 ? "ERROR # ";PEEK(195);" AT LINE ";PEEK(186)+256*PEEK(18
       7):END
VY 400 REM Dataset
CF 410 RESTORE
QG 420 GRAPHICS 0:DIM A$(10)
WC 430 ? "GET MGF. TAPE READY FOR REV.B TO C"
JG 440 ? "AUTORUN FILE AND PRESS RETURN KEY"
CN 450 TRAP 550:INPUT A$
MB 460 OPEN #1,8,0,"C:"
YY 470 READ A:IF A<0 THEN 490
BF 480 PUT #1,A:GOTO 470
GF 490 CLOSE #1:? "*** ALL DONE ***"
NU 500 END
OR 550 ? "ERROR # ";PEEK(195);" AT LINE ";PEEK(186)+256*PEEK(18
       7):END
MI 1000 REM ML data
BI 1010 DATA 255,255,0,6,130,6,169,0,133,2,169,6,133,3,173,250,
        3,240,1
BO 1020 REM DATA 0,1,0,6,14,6,169,60,141,2,211,24,96,96
RN 1030 DATA 169,0,133,216,169,160,133,217,160
JD 1040 DATA 173,1,211,41,253,141,1,211,177,216
IE 1050 DATA 72,173,1,211,9,2,141,1,211,104
HX 1060 DATA 145,216,230,216,208,228,230,217,165,217
FM 1070 DATA 201,192,208,220,162,0,169,12,133,218
```

Action! ★★★★ software

Funkce SIN, COS a SQR

Action! je jedním z nejlepších jazyků vyvinutých pro 8-bitové počítače. Chybí mu však některé funkce, které se ale dají snadno vytvořit. Mezi ty patří i trigonometrické funkce SIN a COS a funkce druhé odmocniny. Protože tyto funkce v programech často využíváme, nabízíme vám způsob, jak s nimi pracovat.

Popsané funkce dovolují bleskové tvoření trojrozměrných výkresů. Příklad tohoto využití uvádíme. Zvláštností funkcí je i to, že nepoužívají reálná čísla díky použití zajímavého triku.

Algoritmus postupu:

1) Natypuj výpis 1 - program FUNKCE.BAS. Zkontroluj správnost pomocí programu TYPO II. Pozor, tento program je v BASICu! Podle toho, zda jsi vlastníkem magnetofonu nebo disketové jednotky, zvol buď řádek 190 (msf) nebo 200 (disk). Na připravené záznamové médium bude po spuštění programu (RUN) nahrán blok dat, který budeme potřebovat pro procedury v jazyku Action!.

2) Zaveď jazyk Action! do počítače. Nyní natypuj výpis 2 - proceduru Insin(), která vkládá do paměti hodnoty funkcí sinus a cosinus a také příslušné funkce SIN, COS a SQR.

3) Nastav cursor na určené místo v proceduře Insin() a potom načti předem získaný blok dat. Načtení provedeme pomocí příkazu CONTROL+SHIFT+R. Takto upravenou proceduru uložíme na disketu nebo na kazetu.

Postup je jasný, dlužíme vám však malé vysvětlení. Argumentem funkcí SIN a COS je úhel vyjádřený ve stupních. Např. A=SIN(45) znamená, že proměnné A je přidělena hodnota sinus 45. Protože jazyk Action! neumožňuje zápis s pohyblivou čárkou, získaná hodnota je 10000 krát větší. Tedy je-li skutečná hodnota sin(45) rovna 0,7071, pak hodnota proměnné A bude 7071. Za upozornění stojí i ta skutečnost, že hodnota funkce není počítaná, ale čtená. To značně zrychluje dobu potřebnou pro získání hodnoty funkce.

Po nezbytném vysvětlení teď přejděme k praktickému využití procedur. Můžeme snadno kreslit kružnice a elipsy, což nám dokazuje výpis 3 - program CIRELI.ACT. Tento program je jen demonstrační. K vlastnímu vykreslení funkcí nám slouží program FUNCTION.ACT (výpis 5). Pro spuštění programu je nutné přidat jeden z několika podprogramů, které jsou uvedeny ve výpisech 4a až 4e. Tvar výkresu bude tedy záležet na zvolené funkci a také na mezních hodnotách x,y,a z. Lze tedy vykreslovat různé funkce. Musíme však dbát na přípustné hodnoty, které platí pro

jednotlivé podprogramy a které požaduje procedura PLot v programu FUNCTION.ACT:

```
     program 4a  ...  -315 <= x <= 470
                      -160 <= y <= 625
                      -100 <= z <= 100
     program 4b  ...  -300 <= x <= 300
                      -300 <= y <= 300
                      -120 <= z <= 220
     program 4c  ...  -300 <= x <= 380
                      -300 <= y <= 300
                       -20 <= z <= 100
     program 4d  ...  -170 <= x <= 170
                      -170 <= y <= 170
                      -100 <= z <= 600
     program 4e  ... -1000 <= x <= 1000
                         0 <= y <= 1000
                      -500 <= z <= 1000
```

Po ukončení vykreslování funkce na obrazovku stiskněte tlačítko START a přejdete do režimu monitoru.

Zatím jsme se zabývali činností a využitím funkcí SIN a COS. Vraťme se však k funkci druhé odmocniny - SQR. Demonstrační program je uveden ve výpisu 6 - program SQR.ACT. Ten dokáže odmocnit čísla v rozsahu 0 až 22000 s přesností 0.5, tedy bez zůstatku. Algoritmus je založen na "ručním" odmocňování. Funkci lze využít v libovolném programu. Můžeme ji tedy připojit i kfunkcím SIN a COS. Pro ilustraci možností tohoto spojení uvádíme program FSQR.ACT (výpis 7), který vytváří graf funkce SQR.

Na závěr uvádíme přehled správně fungujících spojení jednotlivých výpisů:

- program CIRELI.ACT - výpisy 2 a 3
- program FUNCTION.ACT - výpisy 2, 4 a 5
- program FSQR.ACT - výpisy 2,6 a 7

Poznámka: funkci TAN (tangens) dostaneme podílem funkcí SIN/COS. I tu tedy ve svých programech můžeme využít.

Pozor - výpis 1 - program v BASICu je uveden jako poslední. Typování výpisů v jazyku Action! provádějte pozorně, zatím nemáme vyhovující kontrolní program.

Z časopisu BAJTEK přeložil ing. Jeroným Liška
Upravil Jiří Hrdlička

Listing 2 - procedura Insin()

```
; Procedure Insin()
; function SIN and COS

PROC Insin()

  PokeC(88,1536)
  Poke(82,0)
  Position(0,0)
```

```
; zde je třeba načíst blok dat
; vytvořený programem FUNKCE.BAS

  Poke(1543,2)
  Poke(1545,2)
  Poke(1710,2)
  Graphics(0)
RETURN

INT FUNC Sin(INT kat)

  INT m,n

  IF kat<0 THEN kat=-kat n=-1
    ELSE n=1 FI
  WHILE kat>=360
    DO
      kat==-360
    OD
  IF kat>=180 THEN kat==-180 m=-1
    ELSE m=1 FI
  IF kat>90 THEN kat=180-kat FI
  kat=n*mPeekC(kat*2+1536)
RETURN(kat)

INT FUNC Cos(INT kat)

  kat==+90
  kat=Sin(kat)
RETURN(kat)
```

Listing 3 - program CIRELI.ACT

```
; Plot circle and ellipse
; requires Procedure Insin()

PROC Circle(CARD x BYTE y,r1,r2)

INT a,b
CARD d1,d2,i

  di=10000/r1 d2=10000/r2
  FOR i=0 TO 360 STEP 3
    DO
      a=x+Cos(i)/d1
      b=y+Sin(i)/d2
      IF i=0 THEN Plot(a,b)
        ELSE DrawTo(a,b) FI
    OD
RETURN

PROC TestCircle()
```

```
  Insin()
  Graphics(8) color=1
  Circle(160,80,70,20) ;elipsa
  Circle(120,45,30,30) ;kružnice
RETURN
```

Listing 4 - podprogramy pro program FUNCTION.ACT

```
; Trigonometrical Function

; podprogram 4a
; funkce z=sin(x)*sin(x)

CARD FUNC Fun(INT x,y)

INT x1,y1,y2,z

  y1=y*18/31 x=x*18/31
  y2=Sin(y1) x1=Sin(x)
  z=((x1/100)*(y2/100))/100
RETURN(z)

; podprogram 4b
; funkce z=sin(x^2+y^2)/(x^2+y^2)

CARD FUNC Fun(INT x,y)

INT x1,z

  x1=(x/10)*(x/10)+(y/10)*(y/10)
  z=Sin(x1*18/31)/x1
RETURN(z)

; podprogram 4c
; funkce z=cos(x*y)

CARD FUNC Fun(INT x,y)

INT z,xy

  xy=(x*y)/100
  z=Cos(xy*18/31)/100
RETURN(z)

; podprogram 4d
; funkce z=x*x+y*y

CARD FUNC Fun(INT x,y)

INT z
```

```
  z=x*x/100+y*y/100
RETURN

; podprogram 4e
; funkce z=y*Sin(x)/x

CARD FUNC Fun(INT x,y)

INT z

  z=(y/10)*((Sin(x*18/31)/x)/10)
RETURN(z)
```

Listing 5 - program FUNCTION.ACT

```
; 3-D Function Plot
; Requires Procedure Insin()

PROC Plot()

INT zx,sx,zy,sy,zz,sz,a1,a2,x,y
CARD w,j,i,z
BYTE m,n,u

  Print("Xmin=") zx=InputI()
  Print("Xmax=") sx=InputI()
  Print("Ymin=") zy=InputI()
  Print("Ymax=") sy=InputI()
  Print("Zmin=") zz=InputI()
  Print("Zmax=") sz=InputI()
  FOR j=0 TO 512 STEP 2
    DO
      PokeC(15000+j,0) PokeC(15512+j,175)
    OD
  Insin()
  Graphics(24) color=1 SetColor(2,0,0)
  sx=(sx-zx)/17
  sy=(sy-zy)*10/85
  sz=9000/(sz-zz)
  FOR m=0 TO 85 STEP 2
    DO
      y=m*(sy/10)+zy u=0
      FOR n=1 TO 170
        DO
          i=n+m x=n*(sx/10)+zx
          z=Fun(x,y) z=((z-zz)*sz)/100+m
          a1=PeekC(i*2+15000)
          a2=PeekC(i*2+15512)
          IF (a1<a2 OR z>a1 OR z<a2) AND z>m THEN
            IF z<m+90 THEN
              IF u=0 THEN
                u=1 Plot(i+18,170-z)
```

```
                ELSE
                  w=PeekC(91) DrawTo(w+1,170-z)
                FI
                IF z>a1 THEN
                  PokeC(i*2+15000,z) FI
                IF z<a2 THEN
                  PokeC(i*2+15512,z) FI
                ELSE u=0
                IF z<a2 THEN
                  PokeC(i*2+15512,z) FI
              FI
              ELSE u=0
              IF z>a1 THEN
                PokeC(i*2+15000,z) FI
              IF z<a2 THEN
                PokeC(i*2+15512,z) FI
            FI
          OD
        OD
      WHILE Peek(53279)<>6 DO OD
RETURN
```

Listing 6 - program SQR.ACT

```
; Function SQR

BYTE Func Sqr(CARD li)

CARD l,d,c
BYTE a,e,i,w

  i=li a=0
  WHILE l>99
    DO
      a==+1 l==/100
    OD
  FOR i=1 TO 9
    DO
      IF i*i>l THEN EXIT FI
    OD
  w=i-i d=(l-w*w)*100
  WHILE a>0
    DO
      IF a=2 THEN l==*10000 FI
      IF a=1 THEN l==*100 FI
      i=li-l li=l
      IF a=2 THEN l==/100 FI
      c=l+d e=(c/10)/(w*2)
      IF (20*w+e)*e>c THEN e==-1 FI
      d=(c-(20*w+e)*e)*100
      w=w*10+e a==-1
    OD
```

```
  IF (d/10)/(w*2)>4 THEN w==+1 FI
RETURN(w)
```

Listing 7 - program FSQR.ACT

```
; Plot SQR function
; Requires Procedure Insin()

PROC Plot

CARD li
BYTE y,v
INT max,z,p,x,ka

Insin()
Graphics(24) color=1 SetColor(2,0,0)
FOR x=0 TO 127
  DO
    li=16129-x*x y=Sqr(li) max=-32000
    FOR z=-y TO y STEP 3
      DO
        ka=(x*x+z*z)/20 ka=Sin(ka)
        v=ka/400 p=88+v+z/2
        IF p>max THEN max=p
          Plot(160+x,210-p)
          Plot(160-x,210-p)
        FI
      OD
    OD
  WHILE Peek(53279)<>6 DO OD
RETURN
```

Listing 1 - program FUNKCE.BAS Part: 1

```
FT 10 REM ┌──────────────┐
NH 20 REM | Funkce SIN,COS a SQR |
GF 30 REM └──────────────┘
KP 40 REM pro jazyk Action!
YP 50 DIM A$(223):I=1:KAT=0:DEG :FOR J=1 TO 3:RESTORE :GOSUB 15
      0:FOR K=1 TO 31
HI 60 S=INT(SIN(KAT)*10000+0.5):SB=INT(S/256):MB=S-256*SB
HV 70 LI=MB:GOSUB 100:LI=SB:GOSUB 100
PQ 80 KAT=KAT+1:IF KAT=91 THEN GOSUB 150:GOTO 180
HK 90 NEXT K:GOSUB 150:NEXT J:END
HP 100 IF LI<64 OR LI>127 AND LI<192 THEN LI=LI+32:GOTO 120
GE 110 IF LI>63 AND LI<96 OR LI>191 AND LI<224 THEN LI=LI-64
XE 120 IF LI=34 OR LI=155 THEN LI=33
VP 130 IF LI>26 AND LI<32 OR LI>124 AND LI<128 OR LI>155 AND LI
       <160 OR LI>252 THEN A$(I,I)=CHR$(27):I=I+1
EP 140 A$(I,I)=CHR$(LI):I=I+1:RETURN
```

Listing 1 - program FUNKCE.BAS Part: 2

```
IT 150 READ X:IF X=-1 THEN RETURN
CW 160 A$(I,I)=CHR$(X):I=I+1:GOTO 150
QK 170 DATA 32,32,80,114,105,110,116,40,34,-1,34,41,155,-1
RT 180 REM vyber si podle I/O zarizeni
TA 190 OPEN #1,8,0,"C:":REM pro magnetofon
LN 200 OPEN #1,8,0,"D:FUNCTION.DAT":REM pro disk drive
QV 210 ? #1;A$
LC 220 CLOSE #1
```

H E L P

I když je Atari zpravodaj Olomouc zhotovován s použitím výborné 8-bitové techniky počítačů Atari, občas se stane, že selže programová technika nebo uživatel. Stalo se to i nám, proto vás prosíme o prominutí těchto drobných chyb. Mnozí se již u nás informovali, ale stále existuje velké množství uživatelů, kteří tyto potřebné korekce nedostali. Opravte si tedy tyto chyby:

Atari zpravodaj Olomouc 1-2/1989
- str.33, program Polar Plotter II - nedokončený řádek:

```
800 FOR T=1 TO NUM:GOSUB 940+T*10:IF LEN(Q$)>39 THEN
    Q$=Q$(1,39)
```

Atari zpravodaj Olomouc 3-4/1989
- str.25, článek Pianino, chyba v programovém řádku:

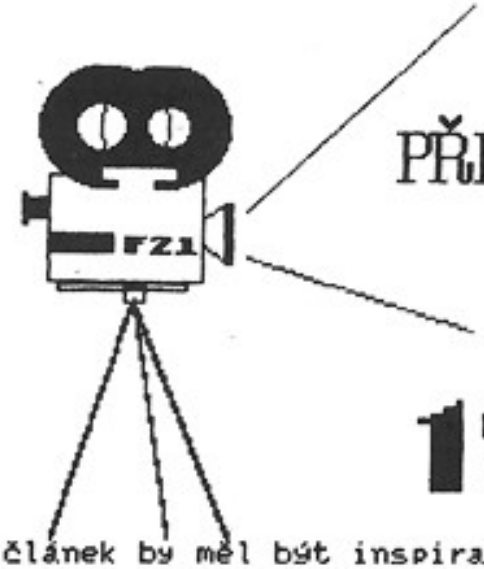
```
2 GOTO 2     (asi uprostřed strany)
```

Děkujeme za dopisy, ve kterých jste nás informovali o těchto chybách, i za náměty, kterými se budeme řídit při výběru dalších článků. Těšíme se na další zajímavé dopisy a na kvalitní příspěvky pro Zpravodaj.

Dokončení programu MLBTOC.BAS

Listing 1 - program MLBTOC.BAS Part: 2

```
SN 1080 DATA 160,0,189,95,6,133,216,232,189,95
KG 1090 DATA 6,133,217,232,189,95,6,145,216,232
WF 1100 DATA 198,218,208,232,165,9,9,2,133,9
TB 1110 DATA 96,223,168,234,224,168,240,225,168,17
UF 1120 DATA 226,168,234,41,187,0,243,191,0,244
SQ 1130 DATA 191,0,245,191,0,246,191,0,247,191
HX 1140 DATA 0,248,191,0,249,191,0,226,2,227
RX 1150 DATA 2,0,6,-1
```



PŘEDSTAVUJEME...

ATARI 192 XT

Tento článek by měl být inspirací pro iniciativního výrobce počítačového příslušenství v Československu. Ukazuje, jak malá polská firma PZ-KAREN provádí profesionální úpravu počítačů značky ATARI 800XL a 130XE, například rozšíření paměti až do 256 kB. Je třeba jen objasnit název ATARI 192XT. Do počítače je totiž zabudován systém s 256 kB paměti, ze kterých se 64 kB používá stejně jako v ATARI 130XE a 192 kB jako paměť přídavná s možností různého použití. Současně s dodatečnou pamětí je instalován modul TurboROM, jakož i interfejs Centronics. Při všech těchto úpravách je zachována kompatibilita s modely ATARI 800/130 XL/XE.

Dodatečná paměť je u ATARI 130 XE rozdělena na bloky po 16 kB, které mohou být volány jednotlivě ze základní paměti počítače v oblasti $4000-$8000. Přepínání bloků paměti řídí registr PORTB (adresa 54017 - hexa $D301). Prostřednictvím odpovídajícího stanovení PORTB je pro bloky 0-3 použit stejně jako v počítačích ATARI 130XE, tzn. nastaví ANTIC do způsobu Extended. ANTIC má ale adresový prostor omezen do velikosti 128 kB.

Nejčastějším použitím takovéhle paměti je RAM-disc, simulující práci stanice pružných disků. Aby bylo možno mít z RAM-disku prospěch, je nutno do základní paměti uložit program obsluhy RAM-disku. Navíc musí mít zapsán v té části paměti, kde nebude kolidovat s ostatními uživatelskými programy a navíc musí odpovědět na volání programů odeslané RAM-disku. Je známo, že v osmibitových ATARI je osazeno plných 64 kB, ale 16 kB této paměti je vyhrazeno paměti ROM s operačním systémem. Příkazem poslaným na adresy $C000-$FFFF se přes MMU (Memory Management Unit) řídí paměť ROM. Při vynulování bitu B0 v PORTB je možno řídit MMU tak, aby řídící bity byly odeslány do nepřípustné RAM. A na tom je založena obsluha RAM-disku. Přepisuje se tudíž operační systém z ROM do RAM, který je pod ním, je uskutečněna změna v obsluze SIO při volání stanice pružných disků a do vybrané oblasti (např. soubor mezinárodních znaků $CC00-$CFFF) se napíše program obsluhy RAM-disku.

Toto má ale některé zásadní vady:
1. Operační systém přepsaný do RAM se lehce může zhroutit
2. Stisknutí RESET přesune systém do ROM a zablokuje přístup do RAM-disku
3. Mnoho programů nemůže v tomto uspořádání vůbec pracovat (TurboBasic, Kyan Pascal, Syn File+, BASIC XE a jiné).

Všechny tyto podstatné vady se podařilo firmě KAREN odstranit. Jak elegantně, to uvidíme dále.

Do počítače byl osazen modul TurboROM. Obsahuje 16 kB paměti SRAM (StaticRAM), do které můžeme v libovolném okamžiku zapsat např. operační systém, tedy přepsat z ROM do SRAM operační systém zabudovaný v počítači a umožnit tak potřebné změny, např. připojit obsluhu RAM-disku, interfejsu Centronics, změny abecedy atd. Po přepnutí přepínače (zabudovaného dodatečně do počítače) je do místa operačního systému, obsaženého v ROM, "podstrčen" systém z paměti SRAM. Přepínač tedy blokuje zápis do této paměti, čili SRAM se chová jako ROM. Výsledkem úpravy je tedy počítač s novým operačním systémem, s prakticky neomezenými možnostmi jeho modifikací, od drobných úprav až po tvoření nových vlastních operačních systémů. Dále se nabízejí tyto další možnosti:

1. Instalace RAM-disku 128 kB, spolupracujícího s DOS 2.5, DOS 3.0, DOS 2.6, DOS 2.3, TOP-DOS, DOS 4.0
2. Instalace RAM-disku 180 kB (Double Density), spolupracujícího s TOP-DOS
3. Instalace obsluhy interfejsu Centronics pro tiskárny standartu EPSON přes porty joysticku
4. Instalace libovolné znakové sady
5. Instalace RAM-disku 128 kB z magnetofonové kazety.

RAM-disky jsou odolné proti RESET. Spolupracují s TurboBASIC, Kyan Pascal, C, ACTION!, BASIC XE, SynFile+, SynCalc, AT Writter, StarTexter aj. Je možno RAM-disk instalovat co by D1:,to tvoří možnost z RAM-disku natahovat bootstrap, při čemž připojíme-li k počítači stanici pružných disků, tak se RAM-disk automaticky přepne na D2. Je umožněna inicializace systému z RAM-disku, což je mnohem rychlejší, než z diskety. Instalace systému do ATARI 130 XE umožní plné využití paměti a současné použití dvou RAM-disků: D2 a D8. Takto přebudovaný počítač umožní perfektní práci. Jednoznačně TurboROM umožní uživateli přizpůsobit systém individuálním představám. Současně s přestavbou počítače se dodává disketa s programy, umožňujícími využití této kvalitní úpravy. Jde o tyto programy:

1. TEST256.OBJ - test paměti. Program umožní provedení testu správné funkce rozšířené paměti. Spouští se z DOSu příkazem LOAD. Obrazovka monitoru se rozdělí na 3 části: okno procesoru ANTIC, okno procesoru 6502 a okno komunikační. V oknech procesorů je zobrazana část paměti mezi adresami $4000-$8000. V procesu inicializace programu se jednotlivé buňky paměti vyplní příslušnými znaky. Při správné funkci vidíme stav bitů PORTB. Uživatel může změnit bity 2-6. Změna hodnoty z 1 na 0 se provede stisknutím příslušné klávesy 2-6

klávesnice, změna z 0 na 1 současným stisknutím CTRL a 2-6. Znaky pro 6502 musí být shodné i v okénku komunikačním.

2. AUTOED.OBJ instaluje RAM-disk 128 kB (Enhanced Density), AUTODD.OBJ instaluje RAM-disk 180 kB (Double Density). Tyto programy umožní instalaci systému v paměti TurboROM, jakož i obsluhu RAM-disku a interfejsu Centronics. Spustí se příkazem LOAD z DOSu, nebo mohou být nataženy jako AUTORUN.SYS. Po natažení se objeví titulek a dotaz: "ED(DD)-RAM-disc as drive 1 or 2" a počítač čeká na určení čísla RAM-disku. Pokud vybereme volbu 1 při připojené stanici pružných disků, automaticky se RAM-disk přečísluje na D2. Po uložení čísla RAM-disku se napíše dotaz "Centronics Printer (Yes or No)?". Volba Y instaluje program obsluhy interfejsu Centronics a umožní připojit příslušnou tiskárnu, která je pak počítačem obsluhována jako běžná ATARI 1029. Tiskárna se připojuje přes porty joysticku. Pak následuje návrat do DOSu nebo Basiku. Uživatel může zapnout TurboROM přepínačem, nainstalovaným na bok počítače. Toto se ihned uvede na obrazovce. Přepnutí systémů je možné kdykoliv.

Upozornění :

a) RAM-disky při instalaci nejsou naformátovány, a proto před kopií souborů na RAM-disk je toto nutno provést

b) Je změněna obsluha některých kláves:

-současné stisknutí SELECT a RESET provede start systému

-současné stisknutí HELP a RESET umožní změnu číslování RAM-disku a pak následuje start systému

c) RAM-disk 128 kB neodpovídá plně disketě naformátované v systému Enhanced. Disketa má 130 kB. V souladu s tím kopírovací programy, např. Sectorcopy 2, nezkopírují sektory 1025-1040. Lépe je použít program Duplicátor 1 nebo 2, který diskety 130 kB správně zkopíruje.

3. DUPLICATOR 1 a DUPLICATOR 2 - sektorové kopírovací programy. Umožní kopírování disket s použitím rozšířené paměti. Tím je při použití jediné stanice pružných disků redukováno přeměňování disket. Spouští se DOSem příkazem LOAD. Diskety se kopírují ve stejné hustotě. Program typu 2 musí mít osazen TurboROM. Po načtení diskety do bufferu má uživatel možnost vykonat start systému, při čemž buffer simuluje stanici pružných disků č.1. Tato funkce se aktivuje klávesou RETURN. DUPLICATOR 2 se znovu spustí klávesou RESET a zkopíruje obsah bufferu na disketu.

4. FCOPY DD - program kopírující soubory. Umožní kopírování disketových souborů s využitím RAM-disku. Po spuštění počítače a instalaci RAM-disku spustíme program FCOPY DD. Kopírujeme z diskety do RAM-disku a z něj na další disketu.

5. FONTLB.OBJ - instaluje nové znakové sady. Jsou k dispozici POLISH.FNT, DM.FNT, ANTIC.FNT, ECKIG.FNT, HOHL.FNT, ELECTRO.FNT.

6. SOUBOR K INSTALACI RAM-DISKU POMOCÍ MAGNETOFONU:

CLOAD.OBJ je natahovací kazetový program, který se použije jako první při použití kazety.

COPYD.OBJ pak umožní nahrát zbývající části systému z kazety. Po jeho spuštění se na obrazovce vypíše listing souborů s písmenovým prefixem. Ke kopírování vybraných souborů na kazetě je třeba pak prefix psát v negativu a stisknout RETURN.

Při tvorbě systémové kazety je třeba kopírovat na kazetu soubory: AUTOCS.BOT, RDEDCS.BOT, CASDOS.BOT, CASDIR.BOT. Tyto programy umožní instalaci operačního systému v TurboROM, jakož i obsluhu RAM-disku. co by D1 a obsluhu interfejsu Centronics. Současně je na RAM-disku uložen program DOS2.5, RAMDISC 130XE, program kopírující v systému disketa - kazeta a naopak - FCOPYCD.OBJ a sektorový kopírovací program v relaci RAM-disc - kazeta - RDCAS. Po načtení systému je vykonán start systému a z RAM-disku se natáhne DOS 2.5. Uživatel pak pracuje stejně jako se stanicí pružných disků.

7. RDCAS.OBJ - sektorový kopírovací program. Umožní kopii vybraných sektorů z RAM-disku na kazetu, jakož i načtení z kazety dříve zapsaného a uloženého souboru zase do RAM-disku.

Tohle by byl v krátkosti obsah diskety, dodávané k systému ATARI 192XT. Jak vidno, pro vytrvalé programátory se otevírá široké pole působnosti a pro ty, kteří stanici pružných disků nevlastní, se vytváří jisté náhradní řešení. Je nutno poznamenat, že pokud se programy z diskety nepoužijí, nemusí se přepínat TurboROM. Práce s počítačem je velmi příjemná. Důležitá je možnost startu počítače bez zničení obsahu RAM-disku. Inicialisace systému trvá 3 sec. a pak je počítač připraven k práci.

Nejdůležitější přednosti ATARI 192XT:
-plná kompatibilita s 800/130 XL/XE
-možnost ustavení RAM-disku po 64 kB nebo jednoho 128 kB
-snadnost přepínání paměti; na disketě je i demonstarční program v BASICu
-možnost bootování z RAM-disku
-unikátní možnost změny v operačním systému, nereagujícím na RESET a studený start systému
-snadnost změny znakových souborů
-možnost obsluhy dalších zařízení přes OS
-možnost užití DOS 2.5 a dvou RAM-disků při práci s magnetofonem
-možnost používání interfejsu Centronics

Redakce časopisu Bajtek měla samozřejmě možnost testu upraveného počítače. Během několikatýdenního používání počítačového systému ATARI 192XL s ním neměli žádné potíže, všechny programy běžely dobře. Redakce proto uvedenou úpravu každému čtenáři může doporučit i na jeho vlastním počítači.

Adresa firmy je:
P.Z.KAREN
ul.Obrońców 23
Warszava
Polsko

Překlad z časopisu BAJTEK:

Markéta Rosípalová
PEGGY Software

Atari XF 551 Disk Drive

Matthew J. W. Ratcliff

Disketová jednotka Atari XF 551 se v současnosti stala jedním z nejvíce zdokonalených a nejvíce diskutovaných výrobků pro 8-bitovou počítačovou řadu Atari. Byla vyrobena, aby nahradila již nevyhovující disketovou jednotku Atari 1050. Před zahájením její výroby se rozpoutala široká polemika mezi uživateli 8-bitových počítačů Atari, kdy někteří uživatelé XF 551 zcela zavrhli, zatímco druzí ji chválili.

Podívejme se nyní na disketovou jednotku Atari XF 551 podrobněji. První věcí, která vás upoutá na XF 551, je to, že je kratší, nižší a širší než drive 1050. Je také neuvěřitelně tichá, což ocení zejména okolí programátora. Na druhé straně je nevhodně umístěn síťový vypínač (zadní strana XF 551, což omezuje uživatele kompaktních "skříňových" systémů), chybí ji indikace připojení k napětí (nemůžete přímo říci, zda je drive vypnutý nebo zapnutý, bez zkoušky jejího přístupu) a je k ní dodáván starý diskový operační systém DOS 2.5. Tento poslední bod vytvořil mnoho zbytečných problémů a stále trvá, mnoho měsíců od zahájení prodeje.

Atari XF 551 je disketová jednotka určená pro diskety typu DS/DD (oboustranné s dvojitou hustotou záznamu). Disketa pak má kapacitu až 360 kB dat, což daleko překonává bariéru 130 kB na mechanismu DD 1050 (při použití zvýšené nebo dvojité hustoty záznamu). Zmatek kolem nové disketové jednotky pramenil z faktu, že dodávaný DOS 2.5 mohl "obsluhovat" jen diskety s jednoduchou (88 kB) nebo zvýšenou (128 kB) hustotou záznamu. Uživatelé tedy předpokládali, že nová disketová jednotka je pouze určitý "klon" staré DD 1050 v novém kabátu. Ale DD XF 551 je schopná obsluhovat disketu s kapacitou 360 kB! Pouze musí mít vyhovující DOS, na který se již netrpělivě čeká. Nový disketový operační systém pro XF 551 byl nazván DOS-XE (pracovní název byl A-DOS). Tento DOS byl vyvinut firmou OSS (Optimized Systems Software), jejíž výrobky jsou dostatečně známy - Action!, BASIC XL-XE, aj. Nový DOS byl zkompletován již před dvěma roky, ale stále není dodáván k nové disketové jednotce. Podle mluvčího fy Atari byl tento nedostatek zaviněn potížemi s tištěním manuálů.

Důvodem uvedení "nové disketové jednotky se starým DOSem" na trh byl spor mezi firmami Atari a Nintendo. Jednalo se o nový výrobek fy Atari - systém XEGS (o něm uveřejníme informace v dalším čísle našeho zpravodaje - pozn. překladatele). Spor se točil kolem reklamního výroku o XEGS, kde se píše, že "připojením disketové jednotky k systému XEGS dostanete plnohodnotný 8-bitový počítač". Reklama nelhala, ale Atari tehdy nevyrábělo disketové jednotky už více jak 6 měsíců. Proto, když se spor registroval, vedení fy Atari rozhodlo o prodeji nové DD XF 551 se starým DOSem 2.5, protože DOS-XE (nebo manuál pro něj) nebyl kompletní. Spor byl zastaven a my dostali konečně novou disketovou jednotku. Problém je však v

tom, že zbytečně plýtváme kapacitou disket. Pokud používáme DOS 2.5, ztrácíme kolem 200 kB kapacity DS/DD diskety!

"Počkejte... Mohu přece obrátit disketu a naformátovat její druhou stranu." O tom jste teď přemýšleli? Bohužel to fungovat nebude. XF 551 používá pouze při formátování časové okénko diskety. Když otočíte disketu, časovací okénko se octne na špatné straně. XF 551 "nevidí" časované pulsy procházející přes okénko diskety, a proto odmítá formátovat obrácenou stranu diskety. To se může zdát přímo nepochopitelné, zvláště když si uvědomíme, že disketová jednotka na nic jiného časové okénko nepoužívá. Když je disketa formátována, časování je zapsáno na disketu. Časovací zařízení je tedy použito jen jako prevence proti formátování obrácené strany diskety. (Řešení je však jednoduché - stačí naformátovat obrácenou stranu diskety na DD 1050. Funguje to spolehlivě, jen je to trochu nepohodlné z hlediska uživatele. Pozn. překladatele).

Toto omezení je však logické, pokud budeme chvíli přemýšlet. Pokud jste používali DD 810 nebo 1050 několik let, pravděpodobně jste formátovali i obrácené strany disket. Najednou začnete používat nový diskový operační systém, který pracuje s hustotou záznamu DS/DD (ovšem předpokládalo se, že budete mít DOS, který toto svede). Nepokazili byste si den, jestliže byste byli nepozorní a naformátovali obrácenou stranu naplněné 360 kB diskety? Ne, protože existuje zařízení, které předchází této vážné chybě.

Výborné vlastností nové jednotky můžeme ocenit, i když zatím očekáváme nový DOS-XE. Použití některých typů DOSů umožní plné využití možností XF 551. Jedná se o MYDOS 4.0, Sparta DOS 3.2 (a výše) a "supercartridge" Sparta DOS X (SDX). (V Československu se rozšířil BiboDOS 6.0 určený pro práci s DD XF 551 - pozn. překladatele.) SDX dokonce plně podporuje zrychlený přenos I/O, který je podobný modifikaci US Doubler pro DD 1050. Zrychlený přenos je standartní vlastnost zabudovaná do disketové jednotky XF 551.

Odborníci firmy Atari přemýšleli dlouho o nové jednotce XF 551. Je příjemně tichá a rychlá, vhodné je i formátování disket typu DS/DD. Například, naformátujeme-li disketu tohoto typu pomocí Sparta DOSu, MYDOSu nebo později DOSu-XE, všechny důležité informace včetně direktory jsou zapsány na straně 1 diskety (strana s etiketou). Po naplnění strany 1 (180 kB dat) začne XF 551 zapisovat data na stranu 2 diskety. Výhoda tohoto systému je zřejmá - můžete si přečíst na jakékoliv jednostranné disketové jednotce s rozšířením hustoty (US Doubler) celý obsah diskety a všechny soubory uložené na straně 1 diskety. Platí to i pro disketu typu DS/SD.

Specifika práce DD XF 551 se stranou 2 diskety popletla již mnoho uživatelů. Pokud disketu naformátujeme jako oboustrannou (DS), můžeme na tento problém zapomenout. DOS bude vybírat stopu podle toho, co je na které straně zaznamenáno. Disketu vždy zasunujte do otvoru disketové jednotky etiketou nahoru (strana 1)! Nikdy nedávejte disketu naformátovanou oboustranně (DS) stranou 2 nahoru - data tam nejsou psána "zezadu", jak si to mnoho lidí myslí. Zmatek vznikl z faktu, že dlouhou dobu jsme používali diskety, které se musely otáčet.

Pokud disketu naformátujeme jako jednostrannou (SS) jakoukoliv hustotou a na obě strany (např. při používání DD 1050 nebo 810), dostaneme dvě zcela separátní diskety. Každá strana diskety obsahuje kompletní direktory a souborovou sadu. Pokud disketu naformátujeme jako oboustrannou, je to stále jedna disketa složená ze dvou stran a jen s jedním obsahem pro všechny soubory na stranách 1 a 2.

Přestože DD XF 551 neumožňuje formátování obrácené strany diskety, neznamená to, že ji neumí číst nebo na ni zapisovat. Musí však být předem naformátována nejlépe na DD 1050. Proto když vlastníte velkou "knihovnu" takto upravených disket, neměli byste s nimi mít vážné problémy.

Používám XF 551 už rok a stále bez vážných problémů. Zakoupil jsem cartridge SDX a dostal tak plné využití disket typu DS/DD s kapacitou 360 kB. To je ideální pro ukládání většího množství dat. Zjistil jsem však, že XF 551 má problémy s formátováním některých "neoznačených" disket (diskety bez firemní etikety). Až dvě z deseti disket jsou špatné. Většinu z těchto disket je možné formátovat na počítačích řady PC, takže se stále uplatní. (To proto, že PC počítače odhalí špatné sektory, zaznamenají je a vyjmou z obsahové mapy diskety. Tak dostaneme kompletně užitečnou disketu. Pro počítače Atari však jeden špatný sektor znamená nepoužitelnost jedné strany diskety.) Diskety bez etiket, pokud se je podaří naformátovat, při pozdějším čtení a zapisování nedělají problémy. Na druhé straně jsem však neměl absolutně žádné problémy s formátováním etiketovaných disket značky Maxell, Verbatim, BASF nebo Polaroid. Doporučuji používat právě tyto diskety, které jsou sice dražší, ale o to kvalitnější.

Jestliže vlastníte program Flight Simulator II nebo některé další komercionální programy s jednoduchým schématem proti kopírování, pravděpodobně nebudou na nové jednotce fungovat. XF 551 pracuje totiž s rychlostí 300 RPM (ot/min), na rozdíl od jednotek 810 a 1050, kde jde o rychlost 288 RPM. Některé z novějších rafinovanějších protikopírujících schémat totiž využívají rychlost otáčení diskety, která byla stanovena na 288 RPM, a přerušují běh programu při porušení této rychlosti. Paradoxem je, že k této situaci došlo po uvedení nového systému XEGS, který tedy nemůže použít novou disketovou jednotku pro čtení scenérií vhodných pro program Flight Simulator II vložený do 128 kB cartridge (dodávaný k systému XEGS). Jinak jsem neměl vážné problémy s odlišnější rychlostí nové disketové jednotky. Zatím jsem neslyšel o žádném programu, s výjimkou FS II, který by nešel spustit na XF 551.

Souhrnně vzato, nová disketová jednotka Atari XF 551 je rychlá, tichá a s kapacitou 360 kB na disketu se řadí k nejlepším výrobkům pro 8-bitové počítače. Už nebude třeba obracet strany disket. Dokud nebude k dispozici DOS-XE, používejte Sparta DOS, MY DOS nebo BiboDOS. XF 551 může dělat rozruch kolem kvality disket, ale to už záleží na každém z vás. Můžete mít také problémy s protikopírujícími schématy, ale to je otázka budoucnosti. Nemožnost naformátovat obrácenou stranu diskety je vlastnost, která vám pomůže zbavit se starých návyků. Pokud jednou začnete používat diskety typu DS/DD, nikdy

se už nevrátíte ke starému způsobu práce, na který jste byli zvyklí při používání DD 1050 nebo 810.

Poznámka překladatele:
Chtěl bych upozornit ještě na jednu informaci, o které se autor článku nezmiňuje. Informace je určena pro ty z vás, kteří si chcete obstarat novou disketovou jednotku přímo z USA. Pokud máte evropský model počítače, disketová jednotka s ním nebude spolupracovat! Disketová jednotka XF 551 vyrobená v USA spolupracuje pouze s počítači vyrobenými tamtéž. Důvodem je odlišný kmitočet, na kterém pracují evropské a americké modely. Cena nové disketové jednotky je také vyšší - pohybuje se od 190 po 290 US dolarů.

Druhá zpráva jistě potěší vlastníky všech těchto disketových stanic. V časopisu ANALOG Computing 9/89 (září) se mezi nabízeným softwarem konečně objevil dlouho očekávaný DOS XE. Pro zájemce uvádím adresu, na kterou se mohou obrátit. Cena programu je 10 US dolarů.

SAN JOSE COMPUTER
The Atari Store
Sunrise Plaza 640 Blossom Hill Rd.
San Jose, CA 95123, USA

ANALOG Computing 2/1989
(c) 1989 by GIA Software
Jiří Hrdlička

FORMÁTY DAT

NA MGF. PÁSCE POČÍTAČE ATARI

Formáty dat na mgf. pásce ATARI

Petr Vičar, AK Rožnov p.R.

1. Úvod

Princip standartního záznamu dat osmibitových počítačů ATARI byl již popsán na několika místech. Chtěl bych se proto v tomto příspěvku věnovat především formátům s jakými se může atarista na magnetické kazetě setkat.

V podstatě jde o dvě nejrozšířenější skupiny. Jedna využívá standartní princip dvoutónové modulace, druhá skupina zcela odlišný způsob fyzického záznamu. Pro osvětlení celého problému bude vhodné uvést přesný postup operačního systému (OS) při zavádění programu z mg.pásky. Pro upřesnění jde o verzi OS z 10.5.1983, kterou jsou vybaveny prakticky všechny počítače tohoto typu v ČSSR.

OS umí zavádět pouze ABSOLUTNÍ formát, v některých pramenech uváděný jako samostartující, nebo také jako kazeta typu "boot". Takováto nahrávka je složena jen ze standartních bloků s touto strukturou:

2 byty	synchronizační (hodnota $55)
1 byte	identifikační (hodnota $FA, $FC nebo $FD)
128 bytů	vlastní data
1 byte	kontrolní součet

Systémová rutina pro čtení bloku předává pouze 128 bytů vlastních dat. Podrobný popis funkce ostatních bytů bloku je uveden např. v olomouckém zpravodaji 3-4/87. První blok absolutního souboru musí mít prvních šest datových bytů co by

hlavičku s tímto významem:

1. indikace ochrany proti kopírování (není-li roven 0)
2. celkový počet bloků v souboru
3-4. zaváděcí adresa
5-6. startovací adresa

Přesný postup OS při zavádění je následující:

(1) Po zavedení všech bloků z mg.pásky do paměti RAM zapíše OS na adresu DOSINI ($C,$D) obsah 5. a 6.bytu hlavičky.

(2) Na adresu RAMLO ($4,$5) zapíše zaváděcí adresu (3.a 4.byte hlavičky) zvýšenou o hodnotu 6.

(3) Provede skok do inicializační rutiny zavedeného programu přes adresu v RAMLO. Pokud v této části nenarazí na instrukci RTS (bez odpovídajícího JSR) , zaváděcí činnost OS tím skončila. V praxi to znamená, že se u některých programů startovací adresa uvedená v hlavičce vlastně vůbec nemusí uplatnit a skutečně tomu tak je.

(4) Jestliže se v části programu podle bodu 3 vyskytne RTS, vrátí se řízení do OS, který otestuje C-bit. Je-li nastaven, ohlásí "BOOT ERROR", v opačném případě provede nepřímý skok přes adresu DOSINI, tedy na adresu startu z hlavičky. DOSINI však může být změněno inicializační rutinou. I zde se opakuje situace s RTS jako v bodu 3.

(5) Pokud se znovu vrátí řízení do OS, zapíše na adresu DOSAKTIV ($9, někde také označovanou jako BOOT?) hodnotu 2. To znamená úspěšné zavedení z magnetofonu. Obsah DOSINI přepíše do CASINI ($2,$3). Další postup bývá zpravidla ovlivněn inicializační rutinou (podle bodů 3 a 4). Standartně skočí nepřímo přes DOSVEC ($A,$B).

Předtím se ovšem pokouší OS zavádět program z diskety a poté i z případného zařízení na paralelním portu.To se u programů, které dospěly až sem, projevuje krátkým zavrčením před startem, stejným jako při spouštění BASICu po zapnutí počítače.

Pokud tedy zaváděný program připustí návrat do tohoto místa OS, musí předem obsadit adresu DOSVEC. Pokud tak neučiní, objeví se SELFTEST, jehož adresa je automaticky po studeném RESETu (vypnutí a zapnutí počítače) zapsána na DOSVEC.

Činnost po RESETU je opět závislá na mnoha okolnostech. Když odhlédneme od takových anomálií, jako je havarijní přepsání některých míst na nulté a třetí stránce RAM, zasunutí cartridge za běhu apod., pak především závisí na obsahu adresy $09 (DOSACTIV). Je-li zde hodnota 2 (viz bod 5), provede se nepřímý skok přes CASINI. Ve valné většině profesionálních programů se během inicializační části programu nastaví

DOSACTIV=1. Po RESETu pak následuje nepřímý skok přes DOSINI.

2. Klasické typy souborů na kazetě

Uvedený typ souboru a jeho standartní zavádění však používá jen malá část programů. Různé softwarové firmy ve snaze po originalitě a často také z důvodů ochrany proti pirátskému kopírování, používají nejrůznější formáty a programátorské triky. Některé z těch, na které jsem narazil, bych zde rád ukázal.

Strojové programy lze z hlediska kódování na kazetách rozdělit do několika skupin:

1. absolutní
2. relativní
3. kilobytové záznamy
4. fast turbo
5. vlastní formát
6. extra formát

Pro všechny je společné využití rutin OS při zavádění z kazety do RAM.

2.1. Absolutní formát

Absolutní formát byl již popsán. Programy v tomto formátu se zavádějí stlačením kláves START+OPTION při zapnutí počítače, nebo je-li počítač v SELFTESTU stačí stisknout a držet START+OPTION a současně krátkodobě klávesu RESET. Pokud podržíme pouze klávesu START, proběhne inicializační rutina a poté se inicializuje BASIC. Této možnosti se používá při zavádění rutin ve strojovém kódu pro jejich další použití v BASICu. Např. nová znaková sada, zařazení nestandartní periferie do OS, změna vlastnosti OS (turbosave) apod.

Výhoda absolutního formátu tedy spočívá v jeho jednoduchém zavádění. Nevýhodou je nutnost dodržení spojitosti logického bloku. Tedy je-li např. zapotřebí zavádět data na adresy $700-$800 a dále pak $2000-$2200, musíme v absolutním formátu zavést spojitý blok dat do $700 až po $2200.

2.2 Relativní formát

Uvedenou nevýhodu odstraňuje formát označený jako relativní, někde též binární diskový. Skládá se z libovolného počtu logických bloků, jejichž velikost a pořadí je nezávislé na fyzických blocích, které opět obsahují 128 bytů platných dat + 4 byty řídící.

Struktura relativního souboru je následující:

/ HLAV1,BLOK1 / HLAV2,BLOK2 /...../ HLAVn,BLOKn/

Každý logický blok má svoji hlavičku dlouhou 4 až 6 bytů.
1. a 2. byte obsahují hodnotu $FF
3. a 4. byte = počáteční adresa bloku (P)
5. a 6. byte = koncová adresa bloku (K)

Za hlavičkou následuje K-P+1 bytů vlastních dat, která budou ukládána od adresy P po adresu K včetně. Počínaje druhým logickým blokem, nemusí hlavička obsahovat 1. a 2. byte, tedy hodnotu $FF.

Tento formát nelze zaváděť přímo službou operačního systému, a proto existuje celá řada tzv. zavaděčů. To jsou v principu krátké absolutní programy, které se zavedou a spustí dříve popsaným způsobem a dále pak řídí zaváděni zbytku programu v relativním formátu. Nejznámější zavaděče z této skupiny jsou BL/C - Binary Load from Cassette, Bootroutine, tzv. vykřičníkový zavaděč, po jehož odstartování se objeví v pravém dolním rohu obrazovky vykřičník, a mnoho dalších. Liší se zpravidla dokonalostí různých kontrol a grafickými efekty. Kontroluje se třeba , zda není vložena Cartridge, zda není současně připojena a aktivní jiná periferie (např. disketa) a podobně.

Společným rysem je, kromě správného zavedení všech bloků na správné adresy, také možnost kdykoliv v průběhu zaváděni spustit určitou část programu. Zobrazí se třeba titulek na obrazovce, spustí se zvukové efekty... Zavaděč kontroluje po načtení každého logického bloku obsah adresy INITAD ($2E2,$2E3). Je-li nenulový, skočí zavaděč nepřímo přes INITAD jako do podprogramu. Přitom není vypnut motor magnetofonu, takže to musí obstarat vyvolaný podprogram, pokud je to zapotřebí. Tedy v případě, že činnost podprogramu je delší než doba, kdy začne magnetofon snímat další fyzický blok dat. K tomuto účelu slouží zřetelně delší meziblokové mezery, které se občas v programech tohoto typu vyskytují. Před návratem do zavaděče musí ještě podprogram vynulovat INITAD. Tato činnost se může během zaváděni opakovat vícekrát.

Konec zaváděni je odvozován buď od statusu "přečten koncový blok", nebo se využívá timeoutu "překročení čekací doby na další vstupní data". Ve většině případů se po ukončeném čtení skočí do programu nepřímo přes adresu RUNADR ($2E0, $2E1). Výjimečně se stává, že zaváděni končí skokem přes INITAD, tedy jakoby do podprogramu, ze kterého se již do zavaděče nevrací.

Z uvedeného vyplývá, že u tohoto formátu je poněkud znesnadněna kontrola správnosti vstupních dat. Záleží na kvalitě zavaděče, ale stává se, že program je sice načten do paměti, avšak je nefunkční, aniž by uživatel identifikoval

vadné místo. S velkým napětím je očekáván výsledek zavádění zejména u programů téměř dvacetiminutových, vybavených zavaděčem, který má potlačené zvukové echo sériového vstupu. To je vrčení, které nás jinak informuje o správně probíhajícím čtení.

Velkou výhodou je možnost zavedení dat na libovolná místa paměti. Délka logického bloku může být od 1 bytu až po hodnotu, která se vejde do operační paměti.

Relativní formát bývá též označován jako "object" a bývá produkován překladači programovacích jazyků. Jsou to např. ATMAS, MAC65, EDITASEMBLER, PASCAL apod.

2.3 Kilobytové bloky

V tomto formátu se k nám dostaly hry jako Fort Apocalypse, Blue Max a mnoho dalších. Vypadá to na pokus o určité urychlení zaváděcí procedury. Programy tohoto typu obsahují speciální zavaděč délky několika stovek bytů a vlastní část programu je rozdělena do fyzických bloků po 1028 bytech:

```
    2 byty hodnoty $55
    1 byte řídící - označení běžného nebo koncového bloku
 1024 bytů = 1 kbyte skutečných dat
    1 byte kontrolní součet
```

Zavaděč používá rutinu OS, "čti blok", která od zadané adresy uloží zadaný počet bytů z magnetofonu. Protože ukládá i první tři řídící byty, probíhá ukládání od vyšších adres, takže data dalšího bloku přepíší tři uložené řídící byty předchozího bloku. Kontrolní součet se do paměti neuloží, pouze se jím otestuje, zda souhlasí se skutečným součtem přeneseného bloku

Schema zavádění:

Úspora není příliš velká, činí 7 meziblokových mezer na jeden kilobyte, t.j. cca 6 sekund. Program však působí kompaktněji a jeho zavádění bývá doprovázeno zobrazením počtu zbývajících bloků.

Kopírovat lze pomocí programu CASDUP jako nestandartní formát.

2.4 Fast turbo

Tento formát je produkován kopírovacím programem Turbotape, který se u nás objevil v holandské verzi. Každý program má na začátku dva standartní bloky svého zavaděče a zbytek tvoří jeden souvislý blok dat s jedním bytem kontrolního součtu na konci. Tady už úspora místa na pásce a času při zavádění činí téměř 1/3 proti standartnímu formátu. Bohužel spolehlivost je díky jedinému kontrolnímu bytu na několik tisíc bytů dat podstatně nižší.

Kopírování neumožňuje žádný ze známých kopírovacích programů. K převodu do "slušnější" formy lze použít program SUPERMON V2.1, ovšem při jeho dobré znalosti.

2.5 Vlastní formát

Pod tímto názvem jsou shrnuty programy, které mají svůj vlastní zavaděč délky 2 až 15 stand. bloků, a zbytek je ve zvláštním formátu. Nelze ho zavádět přímo ani žádným "standartním zavaděčem", tím myslím BL/C apod. Fyzické bloky jsou ale standartní, takže lze tyto programy kopírovat, např. programem CASDUP. Zavaděče v některých případech dekódují vlastní program před jeho uložením do paměti. Mezi zavaděčem a vlastním programem nemusí být delší pauza! Totéž platí i o kilobytových blocích a formátu fast turbo. To, že v souboru vůbec nějaký zavaděč je, poznáme z hlavičky. Počet bloků uvedený ve druhém bytu, nesouhlasí se skutečným počtem bloků na kazetě.

2.6 Speciální formáty

Do této kategorie řadím všechny tzv. nekopírovatelné programy, tedy soubory s takovým formátem, který nelze kopírovat žádným z běžně dostupných kopírovacích programů. Také tyto programy musí mít na začátku nějaký zavaděč. A ten je vlastně klíčem k dalším datům.

Např. systémová rutina pro čtení bloku předpokládá tento formát:

/$55/$55/r/.....n-bytů...../s/

délka bloku může být různá, ale podmínkou pro systém je zachování prvních dvou bytů (hodnota $55 pro synchronizaci záznamu a tedy korekci přenosové rychlosti). Dále řídící byte, jehož hodnota musí být jednou ze tří možností. Poslední byte musí být kontrolním součtem všech předcházejících bytů. Když porušíme některou z uvedených podmínek, nelze takovýto záznam přečíst žádným standartním programem, ale jedině speciálním programem, který si sám řídí čtení z magnetické pásky. Většinou

se tedy setkáme v zaváděči se změnou přerušévacího vektoru VSERIN a zaváděč pak obsahuje modifikovanou rutinu pro vstup jednoho znaku z magnetofonu.

3. Systémy zrychleného nahrávání

Bouřlivý vývoj v této oblasti dospěl od původních 600 Baudů až k desítce kilobaudů a těžko předpokládat, kde se zastaví. Teoretická hranice při použití současných nejkvalitnějších přístrojů a materiálů se odhaduje na 40 kBaudů. Praktický smysl takové záznamové rychlosti přichází v úvahu zejména při zálohování obsahu diskových pamětí.

Zrychlující systémy je možné rozdělit z hlediska uživatele na dvě skupiny. První používá čistě softwarové prostředky, druhá vyžaduje úpravu datasetu.

3.1 "Klasické" turbo

OS obsahuje při čtení z magnetofonu rutinu, která si sama určí přenosovou rychlost. Rozmezí, ve kterém tento mechanismus funguje, je zhruba 400 až 1600 baudů. Spolehlivost však od 800 baudů prudce klesá. I tak je nutné před každým použitím nahrát z magnetofonu modifikovaný ovladač pro kazetu. Druhá možnost je modifikovat přímo operační systém po jeho překopírování do RAM. Touto metodou dosahují autoři slovenského turba spolehlivých 1320 baudů.

Dekodéry ve firemních datasetech jsou však na takové úrovni, že přenositelnost klasického záznamu je problematická už při přenosové rychlosti 800 baudů.

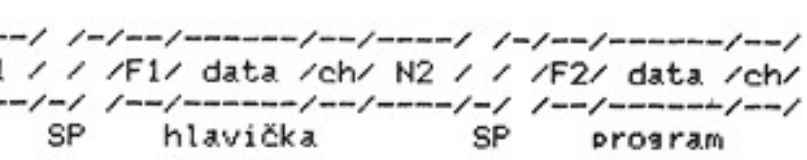
3.2 Turbo ZX

Skutečné řádové zrychlení přináší použití zcela odlišného principu záznamu. Celá skupina těchto turbosystémů vyžaduje úpravu datasetu, spočívající v podstatě ve vyřazení zabudovaného dekodéru dvoutónové modulace. Signál se odebírá ze snímací hlavy, zesílí a natvaruje se a takto se přivádí přímo na vstup počítače.

Formát ZX je v principu shodný se záznamem dat na magnetofon u počítače ZX Spectrum. Jeho podstatou je záznam každého bitu ve formě jednoho úplného pulzu. Přitom délka pulzu pro logickou jedničku je přesně dvojnásobná proti délce pulzu logické nuly.

Každý soubor se skládá ze dvou souvislých bloků. První je hlavička o standartní délce 18 bytů a druhý je vlastní blok dat, jejichž význam je dán hlavičkou. Před každým blokem je nahrán zaváděcí kmitočet (minimálně dvě sekundy pro rozběh

motoru) a synchronizační pulz, který určuje začátek bloku. Před prvním "užitečným" bytem bloku je jeden příznakový byte , který určuje, zda se jedná o hlavičku (=$00) nebo data (=$FF). Formát záznamu je následující:



N1 - návěští 5 sec
F1 - flag hlavičky ($00)
ch - checksum (konrolní suma)
N2 - návěští 2 sec
F2 - flag data ($FF)
SP - synchronizační pulz

a) návěští fZ=808 Hz (619+619 us)
b) synchronizační pulz 190+210 us
c) datový bit L fL=2049 Hz (244+244 us)
d) datový bit H fH=1024 Hz (488+488 us)

Formát hlavičky:

$00 data 1 Typ dat: 0 - program BASIC
1 - numerické proměnné
2 - alfanumerické znaky
3 - blok paměti (CODE, SCREEN$)
$01 data 2
- - - > Název souboru (10 znaků)
- - -
$0A data11

$0B data12 LSB délka souboru
$0C data13 MSB
typ dat
$0D data14 LSB 0 - startovací adresa
$0E data15 MSB 1 - MSB=proměnná nebo 64
2 - MSB=proměnná nebo 128
3 - počáteční adresa

$0F data16 LSB pouze pro typ 0:
$10 data17 MSB délka programu v BASICu

Rychlost záznamu cca 1500 bitů/sec

LSB - nižší byte 16.bitové hodnoty
MSB - vyšší byte - " -

Údaje v předchozím popisu jsou platné pro standartní záznam na počítači ZX SPECTRUM. Na počítačích ATARI se dosud vyskytují následující typy - hodnota 1.bytu hlavičky:

03 - absolutní soubor
04 - relativní soubor (viz kapitolu 2.2)
FE - program pro BASIC - autostart
FF - program pro BASIC

Pro všechny typy je délka souboru ve 14. a 15.bytu hlavičky. Pro typ 03 je ve 12. a 13.bytu zaváděcí adresa, v 16. a 17.bytu startovací adresu.

Při dalším zrychlování jsme dospěli k hodnotám podle tabulky. Označení rychlostí je shodně použito v programech ZXL, SUPERMON, LOADUP, AUTOCOPY.
Rychlost 0 odpovídá formátu ZX SPECTRUM.
Rychlost 1 je totožná s formátem TURBO 2000.

Tabulka frekvencí a rychlostí formátu ZX

rychlost	Lz / us	LS / us	LH / us	LL / us	fz / Hz	fL / Hz	v / Bd
0	619	200	488	244	807	2050	1500
1	372	120	293	148	1340	3380	2500
2	309	100	244	122	1620	4100	3000
3	237	74	183	91	2110	5490	4000
4	196	62	152	76	2550	6580	5000
5	155	50	122	61	3225	8200	6000

Význam použitých zkratek:

us - mikrosekunda
Hz - Hertz
Bd - Baud (bit/sec)
fz - frekvence zaváděcího kmitočtu
Lz - délka pulzu zav. km.
fL - frekvence záznamu logické nuly
LL - délka pulzu log. "0"
LH - délka pulzu log. "1"
LS - délka synchronizačního pulzu
v - průměrná přenosová rychlost

4. Závěr

Článek určitě nevyčerpal vše, co se na kazetách pro ATARI může objevit. Měl by však čtenáři poskytnout přehled těch formátů, které byly nebo jsou v současné době nejrozšířenější. Z tohoto výčtu bylo vypuštěno slovenské turbo 4000, kterému bylo věnováno celé dvojčíslo 5-6/88 olomouckého zpravodaje.

MiSe - program TEXTGR.BAS

Jedná se o krátký podprogram, který umožňuje psát text v grafice 8 (případně v grafice 6 nebo 14). Strojová procedura je uložena od adresy 1536 ($600).

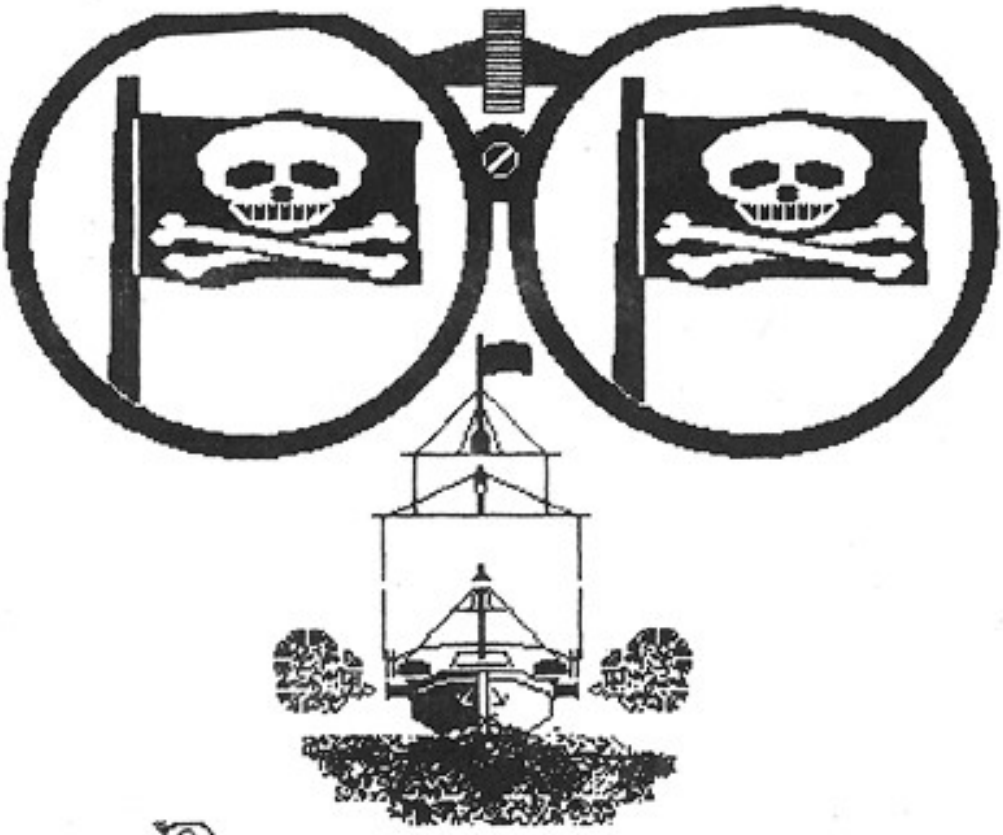
Pomocí příkazu PLOT x,y se nastaví levý horní roh textové řady (řádek 50). Lze tedy zároveň psát text i kreslit obrázky. To vše v jazyku Atari BASIC.

Program byl převzat z publikace Dittrich Clausdorf: Tips Tricks fur Atari a upraven na počítači Atari 800 XL.

Petr Suchý, AK Šumperk

Listing MiSe - program TEXTGR.BAS Part: 1

```
WY 10 REM ┌──────────────────┐
ZK 20 REM │ TEXT - GRAPHICS 8 │
XQ 30 REM └──────────────────┘
FA 40 REM - for all ATARI XL/XE
OW 50 REM - demo program 110-160
WK 60 REM - main program 1000-1060
QR 70 REM (c) 1987
CA 80 REM by Dittrich Clausdorff
TD 90 REM Tips & Tricks fur Atari
EB 100 REM Demo
XW 110 IF PEEK(1536)<>62 THEN GOSUB 1000
UH 120 GRAPHICS 8+16:COLOR 1
OM 130 OPEN #1,8,0,"G:"
DC 140 PLOT 90,80:DRAWTO 230,80:DRAWTO 230,110:DRAWTO 90,110:DR
      AWTO 90,80
WG 150 PLOT 100,90:? #1;"TEXT V GRAFICE 8"
OG 160 GOTO 160
QC 1000 REM Main routine
DG 1010 FOR I=1536 TO 1642:READ A:POKE I,A:NEXT I:POKE 1535,104
      :X=USR(1535):RETURN
LN 1020 DATA 162,0,189,26,3,240,5,232,232,232,208,246,169,71,15
      7,26,3,169,28,157,27,3,169,6,157,28
HA 1030 DATA 3,96,103,6,103,6,103,6,39,6,103,6,103,6,201,155,24
      0,60,56,233,32,133,216,169,0,6
YI 1040 DATA 216,42,6,216,42,6,216,42,9,224,133,217,165,100,133
      ,218,165,101,133,219,162,0,160,0,177,216
IM 1050 DATA 129,218,165,218,24,105,40,133,218,165,219,105,0,13
      3,219,200,192,8,208,234,230,100,208,2,230,101
EO 1060 DATA 160,1,96
```



PIRATE ADVENTURE

PIRATE ADVENTURE

(popis hry)

Tato hra je jednou z řady textových her pro počítače ATARI. Základem každé takové hry je tzv. komunikační jazyk INGLISH. Jde o zjednodušenou formu angličtiny.

Základní slova většiny textových her jsou tyto:

INVENTORY - inventář (seznam předmětů v držení)
QUIT - opuštění hry
SCORE - zatím dosažené body
HELP - pomoc v situacích
(klíčové problémy však musí hráč řešit sám)
TAKE - vzít předmět
DROP - pustit předmět
EXAM - prozkoumat předmět, objekt
GO - jít kam (název světové strany, místo)
OPEN - otevřít (knihu, dveře, atd.)
READ - číst (vzkaz, knihu, atd.)

A nyní již k vlastní hře. Předem bych chtěl upozornit, že k úspěšnému zvládnutí hry je potřeba aspoň pasívní znalost angličtiny a slovník.

Cílem hry je nalézt dva poklady JOHNA SILVERA a donést je do výchozího pokoje. Vzhledem k tomu, že hra vyžaduje logické myšlení, uvedu pouze několik rad, které mohou být užitečné v dílčích problémech.

Hřebíky se dají vytáhmout kladivem .
Rovněž na dřevěnou bednu je vhodné kladivo.
Prázdnou láhev na vodu získáš tak, že rum dáš vypít pirátovi.
Posádku přivoláš tím, že ji probudíš (wake pirat).
Krokodýl má rád ryby.
Ryba potřebuje k transportu slanou vodu, jinak zmizí.
Ve vodě lze plavat pouze s ploutvemi (water wing).
Odemkneš pouze klíčem (unlock ...)
Před vyplutím vytáhni kotvu (weigh anchor).
Kopat lze pouze s lopatou (dig - kopat).
Dobře si přečti mapu a poslechni ji.
Pořádně prostuduj plány.
Pamatuj, že většina obrazů má více východů.
Papoušek má rád sušenky a vůbec se nebojí hadů.
Nechce-li pirát nastoupit na loď, lze vykopat láhev s rumem. (Vypije-li jej, usne.)
Úzkou soutěskou se protáhneš pouze s menšími předměty.
Zkus říct kouzelné slovo YOHO, jestliže máš knihu (say yoho).
I v případě, že zahyneš, Ti může kouzelné slovo pomoci (musíš mít knihu).
Proti uklouznutí jsou nejlepší boty.
Bez zápalek lampu nerozsvítíš.
Pirát na své lodi nesnáší čarovné předměty.
Kotvu v písku lze vykopat.
Za knihovnami někdy bývají tajné prostory.

Tyto rady by Ti měly být nápomocny ve Tvé cestě za poklady. Nebudeš-li si snad vědět rady, obrať se na adresu:

Martin Kubečka
Loučka 200
Nový Jičín
PSČ 741 01

Mnoho štěstí ve hře a dobrou zábavu Ti přeje:

(c) 1989 Martin Kubečka

NESSIE

Tom R. Halfhill

Nessie je akční hra, ve které se na chvíli stanete vědcem-amatérem, který zkoumá tajemství jezera Loch Ness ve Skotsku. Vaším cílem je získat kvalitní fotografii tajuplné příšery, která již delší dobu úspěšně uniká nástrahám, které ji kladou vědci z celého světa. Budete úspěšnější?

Hra funguje na jakémkoliv 8-bitovém počítači Atari XL/XE s minimální pamětí 16kB; s disketovou jednotkou 24kB paměti RAM. Ke hře je potřebný jeden joystick. Hra je vhodná pro všechny věkové skupiny.

Po desetiletí se již táhne spor kolem tajemství jezera Loch Ness. Fanoušci a fanatici, kteří věří na existenci skotské "Lochnesské" příšery, ji zamilovaně nazývají "Nessie". Odtud

tedy pochází název hry. Samotná hra byla inspirována televizním dokumentárním filmem o jezeru Loch Ness, ve kterém byly ukázány desítky fotografií z více než stovky pokusů vyfotografovat tajemné monstrum. Ačkoliv všechny tyto pokusy skončily neúspěšně, vždy se objevily některé podivuhodné okolnosti. Existuje totiž několik kontroverzních fotografií, na kterých můžeme vidět části ploutví, šedé stíny nebo skvrnité obrysy tajemných živočichů. Naše hra simuluje některé z obtíží, se kterými se setkávají vědci i lovci senzací, a ukazuje, jak je těžké vyfotografovat Nesii.

Hra začíná

Když typujete program Nessie (listing 1), vynechejte všechny instrukce REM, pokud váš počítač má jen 16kB RAM. Majitelé disketových jednotek potřebují minimálně 24kB RAM. Natypovaný program zkontrolujte pomocí programu TYPO II.

Po spuštění programu příkazem RUN si počítač vyžádá několik sekund pro inicializaci. Během této doby jsou chráněny speciální oblasti paměti, je nastavena P/M grafika, jsou připravena hrací pole (obrazy) a do paměti jsou uloženy všechny rutiny ve strojovém jazyku (program intenzívně využívá možnosti ML rutin, jak si později ukážeme). V polovině této "čekací" doby se na obrazovce objeví rámeček hledáčku fotoaparátu se zaměřovacím křížem.

Po dokončení celé inicializace se objeví název hry a menu, ve kterém najdete nastavení volby obtížnosti a popis možností získávání bodů pro skore hry.

Hra má dvě úrovně obtížnosti. Přepínání stupně obtížnosti nám zajišťuje tlačítko SELECT. Vybíráte si zaměřovací optiku vašeho fotografického přístroje. Hra má na začátku nastavenu nižší úroveň obtížnosti. Do vašeho fotoaparátu je zamontován širokoúhlý objektiv, kterým zachytíte na fotografii větší oblast. To je indikováno velkým hledáčkem na obrazovce. Po stisknutí tlačítka SELECT přepneme obtížnost na vyšší stupeň. Ve fotopřístroji vyměníte širokoúhlý objektiv za teleobjektiv. Ten je na obrazovce reprezentován mnohem menším okénkem hledáčku. Faktem zůstává, že hledáček teleobjektivu stěží zarámuje Nesii. Optika teleobjektivu je tedy mnohem náročnější na rychlost a přesnou ruku než používání širokoúhlého objektivu. U programu je vše vyřešeno aktuální velikostí hledáčku pro každý ze dvou typů zaměřovací optiky.

Protože používání teleobjektivu je těžší, dostáváte více bodů. Na dolní polovině obrazovky si můžete přečíst, kolik bodů obdržíte za každý možný obrázek, který se vám podaří "vyfotografovat". Například při použití širokoúhlého objektivu čistá, přesně zarámovaná fotografii Nessie obdrží 200 bodů; snímek, ve kterém jste "ořezali" část Nessie, získává 100 bodů; jestliže se spletete a vyfotografujete rybu nebo úhoře místo příšery, dostanete jen 50 bodů; a když vám vyvolají rozmazanou fotografii, za to je 0 bodů. (Určitě se budete v kanceláři tiskové agentury cítit lépe s kvalitním snímkem celé obludy než s pochybnou fotografií nějaké hřbetní ploutve.) Stejné bodové ohodnocení (0 bodů) obdržíte za snímek prázdné hladiny.

Když používáte teleobjektiv, všechny tyto body jsou násobeny koeficientem 10. Program také boduje čas, který potřebujete k vyfotografování celého filmu. Čím déle vám to trvá, tím nižší je výsledné skore.

Po vybrání zaměřovací optiky začíná hra stiskem tlačítka START. Tím se spustí časomíra a zobrazí se hlavní hrací pole.

Kvalitní fotografování

Hra začala. Na TV obrazovce vidíte hlavní hrací pole. Nahoře si všimněte počitadla snímků. To je důležité, protože vám ukazuje, kolik nepoužitých snímků máte ve fotoaparátu. Začínáte s kazetou na 20 snímků. Po každém stisknutí spouště se počitadlo snímků automaticky sníží o 1.

Hledáček vašeho fotopřístroje je na začátku hry umístěn ve středu obrazovky, která reprezentuje jezero Loch Ness. Hledáčkem můžete pohybovat pomocí joysticku libovolným směrem. Stiskem tlačítka FIRE na joysticku se aktivuje spoušť fotoaparátu. Rámeček hledáčku je modrý se zeleným zaměřovacím křížem ve středu. K získání správně centrovaného snímku musíte umístit zaměřovací kříž na Nesii. Když se jakákoliv část Nessie dotkne rámečku hledáčku (při současném stisku spouště), snímek bude na konci hry vyvolán jako rozmazaný. Snímek části Nessie je lepší než nic - to proto, že za něj obdržíme 100 nebo 1000 bodů - ale zdaleka se nepřibližuje hodnotě snímku celé obludy

Z tohoto důvodu musíte být opatrní při fotografování Nessie, aby se vám do hledáčku fotografického přístroje spolu s ní nedostal žádný jiný objekt. Není to tak jednoduché, jak to zní. To konečně poznáte sami. Když se na obrazovce objeví jezero Loch Ness, zjistíte, že je plné ryb a úhořů různých tvarů a barev. Pokud se vám podaří vyfotografovat některé z těchto živočichů místo Nesie, budete zklamáni výsledným bodovým hodnocením - za vaše fotografii získáte pouze 50 nebo 500 bodů (podle stupně obtížnosti).

Důvod pro obdržení aspoň několika bodů je prostý - můžete přece tyto fotografie zaslat do přírodovědeckých časopisů. Ale zpět ke hře. Zvlášť nepříjemní jsou úhoři. Až zlověstně se jejich podoba shoduje s Nesií, což je také důvod, proč tak mnoho zapálených amatérských fotografů má za několik let chuť se vším praštit. Jen profesionálové vytrvají...

Dejte si také pozor na bláznivý pohyb příšery. Nessie není jednoduchý cíl - příšera se objevuje na libovolném místě jezera, plave v nečekaných směrech a potopí se během několika sekund kdekoliv a kdykoliv se jí zachce. Vy mezitím zkoušíte umístit příšeru do středu hledáčku. Když stisknete spoušť a zároveň pohybujete hledáčkem, získáte rozmazaný obrázek. A to pro vás znamená nulové bodování.

Když se "procvakáte" k posledním pěti snímkům filmu, rámeček hledáčku automaticky změní barvu z modré na světle žlutou. Berte to jako upozornění v případě, kdy jste příliš zaměstnáni fotografováním a nedáváte pozor na počitadlo snímků.

Vyvolávání filmu

Po zachycení posledního snímku všechno na chvíli "zamrzne". Potom se "vyčistí" obrazovka a začne proces vyvolávání filmu. Protože slabý záblesk světla v černé komoře by všechno zničil, obrazovka během tohoto procesu ztmavne. Po několika sekundách se objeví výsledné obrázky - podobně jako při vyvolávání se budou fotografie postupně objevovat a zjasňovat.

Každý z 20 snímků ukáže, co jste vyfotografovali, když jste tiskli spoušť. Jsou uspořádány po řadě tak, jak jste fotografovali a každý je opatřen titulkem (mimo ty snímky, které jsou prázdné). Dole na obrazovce se objeví vaše konečné bodové hodnocení upravené podle času, který jste potřebovali.

Ke znovuspuštění hry stačí stisknout tlačítko FIRE na joysticku. Vrátíte se tak zpět k výběru optiky, kterou můžete změnit, než opět začnete hrát.

Poznámky programátora

Nessie je rychlá hra citlivě reagující na podněty, protože nejkritičtější část hry - animace (pohyb hledáčku objektivu) je napsána zcela ve strojovém jazyku (ML) počítače. Rutina ML, která vyplňuje většinu ze šesté strany paměti (1536, $600), konstantně "čte" hodnoty joysticku a podle toho pohybuje hledáčkem. Všechno probíhá během VBI, kdy ve zlomku sekundy se elektronový paprsek vrací z dolní části obrazovky na horní a tak zahájí proces dalšího vykreslení TV obrazu. Protože všechno probíhá 60x za sekundu, pohyb hledáčku je okamžitý, plynulý a bez kmitání.

Vlastní hledáček je vytvořen P/M grafikou. Jsou použity dva objekty hráčů - jeden pro rámeček a druhý pro zaměřovací kříž. Tak může rutina řídící kolize odlišit, zda proběhla kolize mezi Nesií a rámečkem, nebo mezi Nesií a zaměřovacím křížem.

Úkolem 90% BASICu v programu Nessie je inicializace. Když je hra nastavena poprvé, většina z programu se již nikdy nespustí. Použití jazyka BASIC pro tuto práci zjednodušuje program, protože nastavování parametrů ve strojovém jazyku může být velmi pracné. Strojový jazyk byl proto použit jen pro časově náročné operace.

Tento způsob programování se odráží ve hlavní smyčce programu, která začíná na řádku 10000. Je jen 6 řádků dlouhá (a těchto pár řádků bychom mohli ještě zkombinovat a tak dostat ještě kratší smyčku). Protože ML rutina běží automaticky během každého vertikálního zatemnění, opakované volání rutiny příkazem USR z BASICu je zbytečné. Jedinou věcí, kterou BASIC vykonává během hlavní části hry Nessie, je animování ryb, úhořů a přísery. BASIC také kontroluje tlačítko spouště a manipuluje se sekvencí "braní" záběrů (cvaknutí spouště, záblesk obrazovky, uložení hodnot kolizních registrů do pole pro pozdější analýzu). Všechno ostatní provádí strojový jazyk.

V programu spolupracují čtyři ML rutiny. Mimochodem největší je hlavní rutina uložená v šesté straně paměti RAM.

Další rutina zapíná VBI, když hra začíná, a další ji vypíná, když hra končí. Čtvrtá rutina okamžitě přepíná tvar hráče, když je volána instrukcí USR. To je použito při rychlé změně velikosti zaměřovacího hledáčku u té části programu, kdy se rozhodujeme mezi použitím teleobjektivu nebo širokoúhlého objektivu. Velice krátká, ale užitečná rutina.

Předefinované znaky v grafickém modu 2+16 jsou použity pro vytvoření tvaru Nesie, ryb a úhořů během hlavní části hry. Ke zrychlení průběhu animace jsou znaky uloženy přímo do obrazové paměti instrukcí POKE, což je rychlejší než používání instrukcí POSITION a PRINT (při animaci).

Závěrečná část hry, kde probíhá "vyvolávání" filmu, používá modifikovaná display list ke "smíchání" různých grafických modů. Obrazovka je tvořena textovými mody 0 a 1 a to vytváří přijatelný grafický efekt.

Poznámka překladatele:

Program je převzat z knihy COMPUTE!s ATARI Collection (Volume 1), která vyšla asi před pěti lety. V programu jsou použity původní anglické texty a poznámky. Při psaní programu mohou být vynechány instrukce REM, ale potom dávejte pozor při přepisování - nebudou totiž souhlasit kontrolní kody TYPO II. Použité anglické výrazy jsou jednoduché, každý má možnost s pomocí slovníku vše předělat do češtiny.

(c) 1989 by GIA Software
Jiří Hrdlička

"Samozřejmě, to není celá dokumentace.
To je jen index."

Listing 1 - program NESSIE.BAS Part: 1

```
GL 100 REM ┌────────┐
IA 110 REM │ NESSIE │
II 120 REM └────────┘
DN 130 REM - for all ATARI XL/XE
HB 140 REM - BASIC & ML mix
WW 150 REM (c) 1985
FU 160 REM by Tom R. Halhill
KS 170 REM COMPUTE! Publications
EO 180 REM (p) 1989
UM 190 REM by GIA Software
QH 200 GOSUB 11000:REM Initialize
UB 210 GOSUB 12000:REM Redefine characters
XV 215 GOTO 13000:REM Setups
FA 220 GOTO 10000:REM Main loop
ER 1000 REM Move Nessie & decoys
MZ 1020 POKE SCREEN+COORD(OBJECT),0:NOOCOORD=COORD(OBJECT)+MOVE
        (INT(RND(0)*9)+1)
ZQ 1040 IF NOOCOORD<40 OR NOOCOORD>239 THEN COORD(OBJECT)=INT(R
        ND(0)*200)+40:RETURN
BY 1060 POKE SCREEN+NOOCOORD,CHAR(OBJECT):COORD(OBJECT)=NOOCOOR
        D:RETURN
GB 2000 REM Snap photo
SP 2020 POKE 77,0:FILM=FILM-1:POSITION 15,0:? #6;"  ":POSITION
        15,0:? #6;FILM:SOUND 0,0,0,0
DL 2040 FRAME(PHOTO)=PEEK(53252):HAIR(PHOTO)=PEEK(53253):BLUR(P
        HOTO)=STICK(0):PHOTO=PHOTO+1
XE 2060 SETCOLOR 4,9,4:IF FILM<6 THEN POKE 704,28
CR 2080 IF FILM=0 THEN SOUND 0,240,10,15:POP :TIME=INT((PEEK(18
        )*65536+PEEK(19)*256+PEEK(20))/60):GOTO 2200
OP 2100 BUTTON=STRIG(0):RETURN
DK 2200 A=USR(ADR(VBOFF$)):SOUND 0,0,0,0:FOR I=1 TO 1000:NEXT I
        :GOTO 14000
YR 10000 REM Main loop
YW 10020 POKE HITCLR,0
YO 10040 IF STRIG(0)=1 THEN BUTTON=1
CD 10340 IF STRIG(0)=0 AND BUTTON=1 THEN POKE 712,14:SOUND 0,4,
        8,15:GOSUB 2000
KQ 10380 OBJECT=OBJECT+1:IF OBJECT>6 THEN OBJECT=1
QQ 10400 GOSUB 1000:REM Move objects
WA 10420 GOTO 10000
AR 11000 REM Init P/M & ML
GJ 11060 GRAPHICS 2+16:SETCOLOR 2,0,0:? #6;"       nessie":? #6
        :? #6;"    PLEASE  WAIT":? #6;"     21 SECONDS"
RM 11080 PM=PEEK(106)-8:PMBASE=256*PM:HITCLR=53278
IE 11100 FOR I=PMBASE+512 TO PMBASE+768:POKE I,0:NEXT I
JG 11120 RESTORE 11280:DIM WIDEFRAME$(20):FOR I=1 TO 20:READ A:
        WIDEFRAME$(I,I)=CHR$(A):NEXT I:REM Wide viewfinder
NB 11140 RESTORE 11300:DIM TELEFRAME$(20):FOR I=1 TO 20:READ A:
        TELEFRAME$(I,I)=CHR$(A):NEXT I:REM Tele viewfinder
HN 11160 RESTORE 11340:DIM WIDEHAIR$(20):FOR I=1 TO 20:READ A:W
        IDEHAIR$(I,I)=CHR$(A):NEXT I:REM Wide crosshair
RM 11180 RESTORE 11360:DIM TELEHAIR$(20):FOR I=1 TO 20:READ A:T
        ELEHAIR$(I,I)=CHR$(A):NEXT I:REM Tele crosshair
YZ 11200 POKE 704,130:POKE 705,198:REM Blue viewfinder & green
        crosshair
```

Listing 1 - program NESSIE.BAS Part: 2

```
UV 11220 POKE 559,46:POKE 623,1:POKE 53277,3:POKE 54279,PM:POKE
          53256,3:POKE 53257,3:REM P/M setup
OE 11240 HORIZ0=116:VERT0=PMBASE+512+61:HORIZ1=118:VERT1=PMBASE
         +640+66:REM Initial positions
WO 11245 FOR I=1 TO 20:POKE VERT0+I,ASC(WIDEFRAME$(I)):NEXT I:P
         OKE 53248,HORIZ0:REM Draw viewfinder
RK 11250 FOR I=1 TO 20:POKE VERT1+I,ASC(WIDEHAIR$(I)):NEXT I:PO
         KE 53249,HORIZ1:REM Draw crosshair
YZ 11260 REM Viewfinder shapes
VZ 11280 DATA 255,255,129,129,129,129,129,129,129,129,129,129,1
         29,129,129,129,129,255,255,0
PW 11300 DATA 255,129,129,129,129,129,129,129,129,255,0,0,0,0,0
         ,0,0,0,0,0
ZC 11320 REM Crosshair shapes
SM 11340 DATA 16,16,16,16,124,16,16,16,16,0,0,0,0,0,0,0,0,0,0,0
AU 11360 DATA 16,16,124,16,16,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
HH 11380 REM Flip shape routine
XB 11400 DIM FLIP$(25):RESTORE 11420:FOR I=1 TO 25:READ A:FLIP$
         (I,I)=CHR$(A):NEXT I
LN 11420 DATA 104,104,133,204,104,133,203,104,133,207,104,133,2
         06,160,0,177,206,145,203,200,192,20,208,247,96
CC 11440 REM Read joymove routine
LU 11460 RESTORE 11480:FOR I=0 TO 204:READ A:POKE 1536+I,A:NEXT
          I
RS 11480 DATA 174,120,2,224,14,208
FS 11490 DATA 3,32,172,6,224,6
FS 11500 DATA 208,6,32,172,6,32
GO 11510 DATA 107,6,224,7,208,3
KD 11520 DATA 32,107,6,224,5,208
BY 11530 DATA 6,32,140,6,32,107
BV 11540 DATA 6,224,13,208,3,32
KI 11550 DATA 140,6,224,9,208,6
FO 11560 DATA 32,140,6,32,74,6
IN 11570 DATA 224,11,208,3,32,74
DD 11580 DATA 6,224,10,208,6,32
PU 11590 DATA 172,6,32,74,6,76
AH 11600 DATA 98,228,173,253,6,201
RM 11610 DATA 48,240,25,206,253,6
UM 11620 DATA 206,253,6,173,253,6
LK 11630 DATA 141,0,208,206,251,6
RG 11640 DATA 206,251,6,173,251,6
EH 11650 DATA 141,1,208,96,96,173
WA 11660 DATA 253,6,201,176,240,25
ZT 11670 DATA 238,253,6,238,253,6
MR 11680 DATA 173,253,6,141,0,208
WN 11690 DATA 238,251,6,238,251,6
LE 11700 DATA 173,251,6,141,1,208
DM 11710 DATA 96,96,172,252,6,192
WI 11720 DATA 96,176,24,160,0,177
BW 11730 DATA 203,145,205,136,192,0
TP 11740 DATA 208,247,238,252,6,238
XY 11750 DATA 252,6,238,250,6,238
DO 11760 DATA 250,6,96,96,172,252
SM 11770 DATA 6,192,28,144,24,160
```

Listing 1 - program NESSIE.BAS Part: 3

```
WP 11780 DATA 0,177,205,145,203,200
NT 11790 DATA 192,255,208,247,206,252
NX 11800 DATA 6,206,252,6,206,250
WB 11810 DATA 6,206,250,6,96,96
ZM 11820 REM Set up vblank
VT 11900 DIM VBSETUP$(11):RESTORE 11920:FOR I=1 TO 11:READ A:VB
         SETUP$(I)=CHR$(A):NEXT I
YD 11920 DATA 104,162,6,160,0,169,7,32,92,228,96
CB 11940 DIM VBOFF$(11):FOR I=1 TO 11:READ A:VBOFF$(I)=CHR$(A):
         NEXT I
BT 11960 DATA 104,162,228,160,98,169,7,32,92,228,96
LY 11980 DIM LENS$(18):DIM MOVE(9),CHAR(6),COORD(6),FRAME(20),H
         AIR(20),BLUR(20),PICTURE(20),CAPTION(30):RETURN
FR 12000 REM Redefine characters
LL 12020 CHSET=(PEEK(106)-8)*256:FOR I=0 TO 512:POKE CHSET+I,PE
         EK(57344+I):NEXT I
KX 12021 RESTORE 12025
FM 12022 READ A:IF A=-1 THEN RETURN
JA 12023 FOR J=0 TO 7:READ B:POKE CHSET+A*8+J,B:NEXT J
ZW 12024 GOTO 12022
ZJ 12025 DATA 1,255,255,192,192,192,192,129,192
OZ 12026 DATA 3,255,255,3,3,3,3,3,3
NE 12027 DATA 4,3,3,3,3,3,3,3,3
ZN 12028 DATA 5,3,3,3,3,3,3,255,255
NA 12029 DATA 6,0,0,0,0,0,0,255,255
BB 12030 DATA 7,192,192,192,192,192,192,255,255
BO 12031 DATA 8,192,192,192,192,192,192,192,192
EQ 12032 DATA 9,0,0,0,0,0,0,213,127
SN 12033 DATA 10,0,0,64,192,85,127,126,0
VJ 12034 DATA 11,0,0,2,3,170,254,126,0
PC 12035 DATA 12,0,0,0,0,0,0,171,254
LH 12036 DATA 13,0,0,56,125,222,125,56,0
NY 12037 DATA 14,0,0,28,190,123,190,28,0
AZ 12038 DATA 26,5,10,21,42,84,168,80,128
IR 12039 DATA 27,255,255,0,0,0,0,0,0
JP 12040 DATA 32,0,32,48,32,224,224,0,0
FH 12041 DATA -1
ZY 13000 REM Set up screen
UL 13020 SETCOLOR 4,9,4:POKE 756,CHSET/256:? #6;CHR$(125):SETCO
         LOR 3,12,2:SETCOLOR 0,4,8:SETCOLOR 2,0,0
NR 13040 FILM=20:LENS=1:LENS$="wideangletelephoto":SCREEN=PEEK(
         88)+PEEK(89)*256
SO 13060 ? #6;"       NESSIE":? #6:? #6;" <select> ";LENS$(LENS
         ,LENS+8):? #6;"  <start>   game"
XJ 13062 POSITION 0,8:? #6;CHR$(171);"  NESSIE=2000/20000":POSI
         TION 0,9:? #6;CHR$(192);"  CROP  = 100/ 1000"
TO 13064 POSITION 0,10:? #6;CHR$(174);CHR$(172);" FOOLED=  50/
         500":POSITION 0,11:? #6;CHR$(186);"  BLUR  =   0"
KX 13070 POSITION 10,7:? #6;CHR$(171)
QJ 13080 SOUND 0,0,0,0:POSITION 10,3:? #6;LENS$(LENS,LENS+8):IF
          PEEK(53279)=6 THEN 13160
CG 13100 IF PEEK(53279)<>5 THEN 13080
NL 13110 LENS=LENS+9:IF LENS>10 THEN LENS=1
ZN 13115 IF LENS=1 THEN 13125
```

Listing 1 - program NESSIE.BAS Part: 4

```
WE 13120 IF LENS>1 THEN 13140
EL 13125 A=USR(ADR(FLIP$),VERT0,ADR(WIDEFRAME$)):POKE 53256,3:S
         OUND 0,160,10,15
JC 13128 FOR I=VERT1 TO VERT1+2:POKE I,0:NEXT I:VERT1=VERT1+3:A
         =USR(ADR(FLIP$),VERT1,ADR(WIDEHAIR$)):POKE 53257,3
GN 13130 POSITION 9,6:? #6;" ":POSITION 10,7:? #6;CHR$(171):HOR
         IZ1=118:POKE 53249,HORIZ1:GOTO 13080
TF 13140 A=USR(ADR(FLIP$),VERT0,ADR(TELEFRAME$)):POKE 53256,1:S
         OUND 0,80,10,15
SK 13145 VERT1=VERT1-3:A=USR(ADR(FLIP$),VERT1,ADR(TELEHAIR$)):P
         OKE 53257,1
GK 13150 POSITION 10,7:? #6;" ":POSITION 9,6:? #6;"■":HORIZ1=11
         7:POKE 53249,HORIZ1:GOTO 13080
AL 13160 SOUND 0,120,10,15:? #6;CHR$(125):POSITION 0,0:? #6;"■■
         ■nessie FILM=";FILM;"■■■"
OJ 13180 COLOR ASC("■"):PLOT 0,1:DRAWTO 19,1
IQ 13200 MOVE(1)=-20:MOVE(2)=-19:MOVE(3)=1:MOVE(4)=21:MOVE(5)=2
         0:MOVE(6)=19
QO 13220 MOVE(7)=-1:MOVE(8)=-21:MOVE(9)=250:CHAR(1)=9:CHAR(2)=1
         39:CHAR(3)=77:CHAR(4)=201:CHAR(5)=14:CHAR(6)=204
IZ 13240 FOR OBJECT=1 TO 6:COORD(OBJECT)=INT(RND(0)*200)+40:POK
         E SCREEN+COORD(OBJECT),CHAR(OBJECT):NEXT OBJECT
RU 13250 PHOTO=1
JK 13260 POKE 1789,HORIZ0:POKE 1787,HORIZ1
NZ 13270 POKE 1788,61:POKE 203,0:REM VERT0 lo byte
DV 13280 POKE 204,(PMBASE+512)/256:REM VERT0 hi byte
LQ 13290 POKE 1786,194:POKE 207,128:REM VERT1 lo byte
GL 13300 POKE 208,(PMBASE+512)/256:REM VERT1 hi byte
YF 13310 POKE 205,2:REM Lo byte for vert memory shift
XJ 13320 POKE 206,(PMBASE+512)/256:REM Hi byte for vert memory
         shift
QP 13330 IF LENS>1 THEN POKE 1647,192:POKE 1680,106:REM Reset r
         ange check for telephoto
XW 13340 A=USR(ADR(VBSETUP$)):POKE 18,0:POKE 19,0:POKE 20,0:SOU
         ND 0,0,0,0:GOTO 220
JG 14000 REM Develop film
MC 14050 POKE 53277,0:POKE 53261,0:POKE 53262,0
GR 14060 GRAPHICS 0:SETCOLOR 2,0,0:DLIST=PEEK(560)+256*PEEK(561
         ):SCREEN=PEEK(88)+256*PEEK(89):REM P/M off
WJ 14070 POKE 756,CHSET/256:POKE 752,1:? CHR$(125);:POSITION 0,
         16:POKE 82,0
QX 14080 RESTORE 14100:FOR I=6 TO 26:READ A:POKE DLIST+I,A:NEXT
          I
EK 14100 DATA 6,6,6,2,2,6,6,6,2,2,6,6,6,2,2,6,6,6,2,2,6
SP 14110 FOR I=1 TO 15:READ A:POKE SCREEN+672+I,A:NEXT I
IY 14112 FOR I=PMBASE+512 TO PMBASE+768:POKE I,0:NEXT I
QE 14115 DATA 36,37,54,37,44,47,48,41,46,39,0,38,41,44,45
HD 14120 FOR II=1 TO 4
DK 14140 FOR I=0 TO 19:READ A:POKE SCREEN+40+I,A:NEXT I
OA 14160 DATA 1,27,3,0,1,27,3,0,1,27,3,0,1,27,3,0,1,27,3,0
HC 14180 FOR I=0 TO 19:READ A:POKE SCREEN+60+I,A:NEXT I
KR 14200 DATA 8,0,4,0,8,0,4,0,8,0,4,0,8,0,4,0,8,0,4,0
JJ 14220 FOR I=0 TO 19:READ A:POKE SCREEN+80+I,A:NEXT I
UR 14240 DATA 7,6,5,0,7,6,5,0,7,6,5,0,7,6,5,0,7,6,5,0
```

Listing 1 - program NESSIE.BAS Part: 5

```
NP 14260 RESTORE 14160:SCREEN=SCREEN+140:NEXT II
SF 14280 RESTORE 14300:SCREEN=PEEK(88)+256*PEEK(89):FOR I=0 TO
         5:READ A:POKE SCREEN+606+I,A:NEXT I
JH 14300 DATA 46,37,51,51,41,37
QE 14350 FOR I=1 TO 20:READ A:PICTURE(I)=A:NEXT I
XL 14355 DATA 61,65,69,73,77,201,205,209,213,217,341,345,349,35
         3,357,481,485,489,493,497
DH 14356 FOR I=1 TO 30:READ A:CAPTION(I)=A:NEXT I
NK 14357 DATA 34,44,53,50,0,0,35,50,47,48,0,0,38,47,47,44,37,36
         ,46,37,51,51,41,37,0,0,0,0,0,0
JK 14360 RESTORE 14510:SCORE=0:SETCOLOR 1,0,0:FOR I=1 TO 21
MZ 14365 IF I=21 THEN 14520
TM 14370 IF BLUR(I)<>15 THEN POKE SCREEN+PICTURE(I),90:X=0:GOSU
         B 14500:NEXT I
RJ 14380 IF HAIR(I)=4 AND FRAME(I)<>4 THEN SCORE=SCORE+2000:POK
         E SCREEN+PICTURE(I),203:X=18:GOSUB 14500:NEXT I
HQ 14390 F=FRAME(I):H=HAIR(I)
AM 14400 IF F>3 AND F<>8 AND F<>9 AND F<>10 THEN SCORE=SCORE+10
         0:POKE SCREEN+PICTURE(I),96:X=6:GOSUB 14500:NEXT I
GU 14420 IF HAIR(I)>0 THEN SCORE=SCORE+50:POKE SCREEN+PICTURE(I
         ),76:X=12:GOSUB 14500:NEXT I
BF 14440 X=24:GOSUB 14500:NEXT I
TY 14500 SOUND 0,12*I,10,7:READ A:FOR II=1 TO 6:POKE SCREEN+PIC
         TURE(I)+A+II,CAPTION(II+X):NEXT II
DZ 14505 SOUND 0,0,0,0:RETURN :REM Print captions
IW 14510 DATA 38,42,46,50,54,38,42,46,50,54,38,42,46,50,54,38,4
         2,46,50,54,54:REM Caption locations
LZ 14520 FOR I=0 TO 6:SETCOLOR 0,5,I:FOR II=1 TO 8:NEXT II:NEXT
          I
YY 14540 FOR I=0 TO 8:SETCOLOR 3,15,I:FOR II=1 TO 8:NEXT II:NEX
         T I
FN 14550 FOR I=0 TO 10:SETCOLOR 1,12,I:FOR II=1 TO 8:NEXT II:NE
         XT I
DF 14560 RESTORE 14570:FOR I=0 TO 6:READ A:POKE SCREEN+632+I,A:
         NEXT I
RP 14570 DATA 51,35,47,50,37,0,29
DW 14620 RESTORE 14640:FOR I=1 TO 22:READ A:POKE SCREEN+666+I,A
         :NEXT I
SG 14640 DATA 52,47,0,50,37,48,44,33,57,0,51,46,33,48,0,51,40,5
         3,52,52,37,50
VB 14650 IF LENS>1 THEN SCORE=SCORE*10
LG 14660 ? INT((4/TIME)*SCORE);
JI 14680 IF STRIG(0)=1 THEN 14680
GX 14700 SOUND 0,4,8,15:FOR I=1 TO 3:NEXT I:SOUND 0,0,0,0:POKE
         53248,0:POKE 53249,0:POKE 704,130
NT 14720 GRAPHICS 2+16:HORIZ0=116:VERT0=PMBASE+512+61:HORIZ1=11
         8:VERT1=PMBASE+640+66:POKE 53277,3:POKE 559,46
UA 14740 POKE 53256,3:POKE 53257,3:FOR I=1 TO 20:POKE VERT0+I,A
         SC(WIDEFRAME$(I)):NEXT I:POKE 53248,HORIZ0
UB 14760 FOR I=1 TO 20:POKE VERT1+I,ASC(WIDEHAIR$(I)):NEXT I:PO
         KE 53249,HORIZ1:GOTO 13000
```

8-BIT ATARI 8-BIT XL/XE

SOFTWARE PROTECTION TECHNIQUES

Listing 1 - program PROTECT.BAS Part: 1

```
YA 0 POKE 566,158:POKE 580,1
MK 1 TRAP 100
NT 2 DIM X$(40):POKE 764,33
TK 3 GOSUB 10
SB 4 X$="█████H█THD█ █████L1█"
WA 5 X=USR(ADR(X$))
FX 10 POKE 752,1
CJ 15 ? "↰":? :? "Loading, please wait...":RETURN
SG 100 GRAPHICS 0
NU 110 ? :? :? "Error in loading: ";PEEK(195)
YP 120 POKE 580,0:POKE 566,146:NEW
PS 200 REM Variable X$ includes:
WJ 210 REM INV " - ESC CTRL+2 - INV CTRL+Z - INV ) - INV 7
XR 220 REM H - INV ) - T - H - INV ) - CTRL+D - SPACE - INV 6
FE 230 REM INV ; - INV ) - ESC CTRL+INSERT - L - CTRL+D - INV ;
```

Jak chránit program před vylistováním

Občas se stává, že byste chtěli chránit jeden z vašich programů v BASICu před vylistováním na obrazovku nebo tiskárnu. Ať už máte pro ochranu programu jakýkoliv důvod, existuje několik metod, které byste měli znát.

Nejjednodušší metoda je použití tzv. souborů RUN-only (pouze-RUN). Když použijete tohoto způsobu, můžete sice nahrát program do paměti příkazy ENTER nebo LOAD, ale nemůžete ho již vylistovat příkazem LIST. Program se tedy dá spustit pouze přes RUN "D:FILENAME" nebo RUN "C:", kde FILENAME=název souboru.

Jestliže jste připraveni použít tuto metodu, přidejte k programu, který chcete chránit, tento řádek:

```
32767 POKE PEEK(138)+256*PEEK(139)+2,0:SAVE "D:FILENAME":NEW
```

Potom zadejte GOTO 32767. Tím se váš program uloží ve chráněné verzi na záznamové médium, v našem případě na disketu.

Uživatelé kazetových magnetofonů použijí obměněnou verzi

```
32767 POKE PEEK(138)+256*PEEK(139)+2,0:SAVE "C:":NEW
```

Jako v případě disketové jednotky, zadejte příkaz GOTO 32767 k uložení programu na kazetu. Potřebujete k tomu ještě připravený kazetový pásek a stisknuté tlačítko PLAY/RECORD na magnetofonu.

Abyste při běhu programu zabránili opětnému spuštění řádku 32767, přidejte k programu ještě tento řádek:

```
32766 STOP
```

Upozornění na závěr. Pro všechny případy se nechte někde schovanou i nechráněnou verzi programu. Může se vám hodit třeba pro pozdější modifikaci programu.

Pár slov o zajištění programů

O zajištění programů na 8-bitových počítačích Atari se v současnosti hodně mluví a píše. V následujícím příspěvku je uvedeno několik velmi konkrétních rad k danému tématu.

Úvodem si připomeňme obecné zásady ochrany programu napsaného v jazyku Atari BASIC:

1) vypnout z činnosti klávesu BREAK
2) přeprogramovat funkci SYSTEMU RESET
3) zajistit program proti chybám (příkaz TRAP)
4) znemožnit uživateli vylistování programu

Pokud jde o klávesu BREAK, nejlepším řešením je umístění příkazu POKE 566,158 na začátek programu a příkazu POKE 566,146 na konec. Tlačítko RESET nejsnadněji přeprogramujeme pomocí příkazu 580,1 na začátku a příkazu POKE 580,0 na konci programu. O funkci SYSTEMu RESET bylo již dost napsáno, a proto si můžete zvolit libovolnou metodu. Autor však dává přednost zlomyslnému řešení - tzv. "studenému startu" počítače. Ovšem i zlomyslnost má své meze, a proto upozorňuje, že je nutné do

adresové buňky 580 vložit hodnotu 0 na konci programu. Jinak můžeme uživateli způsobit nemilé překvapení v podobě zablokovaného systému. Ochrana proti programovým chybám (většinou chyby vstupu/výstupu nebo hodnot smyčky) závisí na typu konkrétního programu. Zapamatujme si, že v jazyku Atari BASIC existuje instrukce TRAP!

A nakonec čtvrtý problém a současně jádro tohoto článku. Otázka výpisu programu je vlastně nejdůležitější, protože ani přeprogramování klávesy BREAK ani SYSTEMu RESET nepomůže, zadáme-li příkaz LIST před spuštěním programu. Autor knihy "Mapping the Atari", Ian Chadwick, uvádí podprogram, jehož výsledkem je účinné a nezvratné zajištění naší verze programu před vylistováním. Podprogram se připisuje až na konec (tady pozor!) chráněného programu a vyvolává se instrukcí GOTO. Na záznamovém médiu (disketa, mgf. pásek) je pak uložena chráněná verze programu. Vadou této metody (obzvláště pro majitele magnetofonů) je to, že program lze nyní spustit pouze jediným způsobem, a to příkazem RUN "D:FILENAME" nebo RUN "C:". (Popsaná metoda viz předchozí článek.) Kdo by však rád nahrával program mnohonásobně déle, než je nutné?

Proto pro majitele magnetofonů nabízíme novou verzi známého programu, kde příkaz SAVE "C:" nahradíme příkazem CSAVE.

```
NJ 31000 FOR VARI=PEEK(130)+PEEK(131)*256 TO PEEK(132)+PEEK
         (133)*256:POKE VARI,155:NEXT VARI
WF 31100 POKE PEEK(138)+PEEK(139)*256+2,0:CSAVE:NEW
```

Návod k použití:

1) řádky 31000 a 31100 jsou posledními řádky programu
2) pro uložení chráněného programu na záznamové médium použijte příkaz GOTO 31000
3) v programu nesmí být použita instrukce LIST
4) program musí být ukončen příkazem NEW

Takto zajištěný program nebude reagovat ani na RUN ani na LIST ani na RUN "C:". Bude ale výborně fungovat, použijeme-li pro něj speciální zaváděcí program - viz listing 1.

V zaváděcím programu můžete vyhodit komentáře (REM) a případně podle potřeby můžete doplnit podprogram pro titulní stranu (řádek 50). Ve vlastním chráněném programu již nemusíte zadávat příkazy POKE 580,1 a POKE 566,158, i když pro každý případ je lépe, nechat je tam. Také je užitečné "vyhodit" všechny komentáře REM, které jsou nyní zbytečné, protože případné opravy programu lze provádět jen na nezajištěné verzi. Proto pozor, vždy si tuto verzi programu uschovejte! Také si zapamatujte, že program musí být zakončen touto sekvencí příkazů:

```
POKE 580,0:POKE 566,146:NEW
```

Použitá literatura : Atari EXPLORER 1986
BAJTEK 8/1988

ing. Jeroným Liška
Jiří Hrdlička

Obsah Atari zpravodaje Olomouc 5-6/1989:

	Název:	str:
1.	Sparta DOS X	1
2.	Od Kyan PASCAL-u k Turbo PASCAL-u!	2
3.	MiSe - program MLBTOC.BAS	22
4.	Action! - funkce SIN, COS a SQR	24
5.	Help!	31
6.	Představujeme... ATARI 192 XT	32
7.	Atari XF 551 Disk Drive	36
8.	Formáty dat na mgf. pásce Atari	40
9.	MiSe - program TEXTGR.BAS	49
10.	Pirate Adventure - popis hry	50
11.	Nessie	52
12.	Ochrana programů v Atari BASICu	62

Redakční oznámení:

!!! PROGRAMÁTORI - POZOR !!!

Hledáme autory článků - SOFTWARE i HARDWARE 8-bitových počítačů Atari XL/XE. Neváhejte a pošlete článek do naší redakce. Přivítáme i překlady ze zahraniční literatury. Vždy však uvádějte, na základě jaké publikace nebo časopisu jste článek zpracovali. Honorář máme možnost vyplácet v rámci směrnice pro hospodaření ZO Svazarmu.

Od roku 1989 začínáme zpracovávat veškeré informace, které se objeví v našem zpravodaji, na počítačích Atari. Prosíme Vás tedy o následující:

Příspěvky zasílejte pokud možno napsané v programu Čapek (verze 1.1 nebo 2.x) a uložené na magnetofonové kazetě (nejlépe C-60) nebo disku. Raději svůj příspěvek uložte 2x nebo 3x za sebou, totéž platí i v případě programu. Během tří týdnů se Vám kazeta nebo disk vrátí. Značně tím urychlíte průběh zpracování jednotlivých čísel Zpravodaje a pomůžete zlepšit namáhavou práci redakční rady.

Zásilky posílejte na adresu olomouckého Atari klubu a označte je nápisem ZPRAVODAJ.

Naše adresa: Atari klub Olomouc
PS 137
772 13 OLOMOUC

Zpravodaj AK Olomouc 5-6 1989
NEPRODEJNĚ, odběr vázán na příspěvek.

Odpovědný redaktor: Ing. Kopečný Pavel
Odborný redaktor : Hrdlička Jiří

Neprošlo jazykovou úpravou.
Předáno do tisku: říjen 1989

Tisk povolen OK ONV Olomouc, 0380500387
Tisk: MTZ Olomouc

30-j