STRETNUTIE S PASCALOM

1. časť

Určite to poznáte! Prvý zápal z počítača, kedy spúšťate jednu hru za druhou, je zápal očných spojiviek od prebdených nocí nad novozískaným „členom rodiny“. Niektorým sa to tak zapáči, že ostávajú pri hrách už navždy, a tak sa rodia špecialisti na DOOMy a ostatné chuťovky tejto novodobej drogy. Zvyšok prechádza druhou etapou vývoja. Skúšajú rôzne textové a grafické editory, kalkulačné a databázové programy alebo celkom užitočné „kusovky“, ktoré im nakoniec prilnú k srdcu, lebo ich pomerne úspešne používajú pri svojej profesii. (Aj tu sa nájdu rôzni zberatelia, ktorí sú schopní vymenovať desiatky programov toho istého typu, všetky pracne držia na harddisku, ale aj tak píšu nakoniec v tešesťstodvojke...). Drvivá väčšina ľudí z určitých pochopiteľných dôvodov končí na tejto platforme vývoja počítačníkov, pretože im tento stupeň počítačovej gramotnosti stačí. Hovorí sa im užívatelia. Ale z tejto skupiny sa vyčlenia nadšenci, ktorých lákajú počítačové diaľavy. Orientujú sa spravidla iba na jednu úzko vymedzenú oblasť kompjúterového spektra, kde dosahujú vysoko profesionálnych výsledkov. Rôzni grafisti, CADisti, bejzisti (pôvodne som chcel napísať basisti, ale z určitých dôvodov som radšej použil fonetický tvar...aby sa niekto neurazil) a hlavne sieťari sú Páni a Dámy (áno aj Dámy) s veľkým P a D. Bez ich nadšenia a profesionality by neexistovali rôzne efektné reklamy, časopisy, designy strojov (alebo aj šiat) a ani najrozmáhajúcejšia vetva ľudskej elektronickej komunikácie – sietí. Ľudia sú tvory bádavé, a tak časť prejde treťou etapou vývoja. Nestačí im používanie toho-ktorého produktu, chcú sami vytvoriť svoj vlastný program, ktorý si „ušijú“ presne na mieru. Vstupujú na horúcu pôdu programátorskú, nevediac, že je to oblasť plná horkosti a sklamaní, ktoré nakoniec absolútne preváži nepopísatelná radosť z fungujúceho programu. Keď vytvoria produkt, ktorý začne používať určitá skupina užívateľov, získavajú nielen hmotný zisk, ale aj renomé. Vedomie, že užívatelia používajú produkty programátorov, je asi najväščou hybnou silou, ktorá akoby švihnutím čarovného prútika zastavuje bolesti hlavy kvôli prebdeným nociam nad listingami programov, dáva silu do boja s rodinným rozpočtom alebo drobnou nevraživosťou rodiny pre čas, ktorý ste predtým venovali jej. Lebo verte, milé dámy a páni, že o peniaze a čas vás pripraví žena najpríjemnejšie, alkohol najbolestnejšie a počítač najrýchlejšie. Stať sa programátorom nie je otázka rozhodnutia. Je to vec, ktorá chce trochu talentu, chuti a štúdia. Zatiaľ čo prvé dve vlastnosti sú dannosti a prejavia sa okamžite, štúdium vyžaduje čas. So štúdiom súvisí výber prostriedkov, ktorými chceme dosiahnúť svoje programové predstavy a ciele. Ja som si z určitých dôvodov (schválne neuvádzam akých, lebo nechcem rozvíriť nikdy nekončiacu polemiku, či Pascal alebo C) vybral programovací jazyk Turbo Pascal od firmy Borland. Jazyk vhodný ako na výuku programovania, veď bol k tomu zrodený, tak aj na tvorbu veľmi solídnych projektov.

Čo je pascal

Pascal je vyšší programovací jazyk. Vytvoril ho pán Wirth ako pomôcku programovania pre študentov. Vďaka jednoduchému osvojeniu a veľkej modularite sa veľmi rýchlo rozšíril a v určitých implementáciách existuje dodnes a dni jeho zániku aj napriek rôznym predpovediam sú stále v nedohľadnu. To, že je vyšším programovacím jazykom, znamená, že odstraňuje problémy strojovo orientovaných jazykov, kde program bola postupnosť elementárnych inštrukcií bez kontroly správnosti zápisu a činnosti. Vyšší programovací jazyk zavádza vhodné štruktúry, ktoré zvyšujú zrozumiteľnosť napísaných programov, a tým zjednodušujú programovanie. Medzi prvé aj u nás veľmi známe implementácie jazyka pascal patrí program od firmy HISOFT, ktorý fungoval na starom dobrom Sinclairi. Medzi pécečkármi je snáď najznámejšia implementácia Wirthovho pascalu od firmy BORLAND International, ktorej vlastníkom je Phil Kahn. Volá sa TURBO PASCAL a jeho zatiaľ posledná verzia nesie číslo 7. Najobľúbenejšia bola verzia šesť, a keďže sa sedmička v základoch od šesťky nelíši a je dostupná na mnohých počítačoch či už v školách alebo doma, budeme sa venovať predposlednej verzii.

Musíme si uvedomiť, že TURBO PASCAL (ďalej už len TP) je kompiler. To znamená, že prekladač preloží zdrojový text do spustitelného súboru s príponou .EXE, ku ktorého činnosti už nie je potrebný samotný TP. Teda doma v TP napíšem program, po preložení ho prenesiem na počítač, kde žiadny pascal nie je a môžem ho spustiť. Prostrediu samotného TP hovoríme vývojové prostredie.

Čo je Turbo Pascal 6.0

TP 6.0 je programový integrovaný balík, software, ktorý v komplexe obsahuje vývojové prostredie, prekladač, knižnicu podprogramov, rôzne utility, help, vzorové príklady a dokumentáciu (v angličtine).

Dodáva(l) sa v dvoch variantách, a to ako Turbo Pascal 6.0 a Turbo Pascal 6.0 Professional. Niekde na našom počítačovom trhu bol aj český help, ktorý nahradil anglickú verziu.

Čo budeme potrebovať

- TP 6.0 pobeží na ľubovoľnom PC s pamäťou aspoň 1 MB a miestom na disku do 7 MB. (Ja som začínal s TP 5.5 na PP 06 bez harddisku a s 2 mechanikami a CGA kartou. To boli časy...prekladu.).
- Najvhodnejšia je VGA karta a farebný monitor, ale postačí aj EGA alebo monochrom.
- Ďalej myš, ktorá nie je nevyhnutná, ale veľmi spríjemní prácu v prostredí TP.
- Ak máte tlačiareň alebo aspoň prístup k nej, ušetríte si časté omyly. Výpis z tlačiarne sa študuje podstatne lepšie ako výpis na obrazovke, ktorá vám poskytuje stránku o 25 riadkoch. Ale ani tlačiareň nie je nevyhnutná.
- Čo je nevyhnutné, je samotný TP 6.0. (Hmm, tu musím upozorniť na legálnosť programu, hmm, hmm...). Tí, ktorí majú TP 7, môžu pracovať s nami, odlišnosti nebudú.(Zatiaľ).
- Veľmi by sa hodil aj vyššie spomínaný český help, nebol ani drahý, za pár stoviek sa dá získať a stojí to za to. Pre tých, čo sú dobrí v angličtine, to neplatí.

Ako sa učiť programovať

Teórií o výuke programovania sa už napísalo dosť. Každá je v niečom výhodnejšia od druhých, každý vyučujúci si ju upraví podľa svojej predstavy. Metóda, keď sa preberá príkaz TP abecedne jeden za druhým, je asi tá najmenej šťastná.

Začínal som tak, že som dostal zdrojový text hotového fungujúceho programu. Ten som si po preložení vyskúšal, zistil som, čo vlastne robí. Potom som program krokoval a sledoval jeho činnosť. V dostupnej literatúre som naštudoval konkrétny príkaz jednotlivého kroku, poprípade zmenil jeho parametre. To viete, že som spravil aj také zásahy, kedy už program nefungoval vôbec. Ale vďaka odloženému originálu a posledným zmenám som sa mohol vrátiť o jeden-dva kroky späť a skúšať znovu.

Zdá sa vám to ťažkopádné? Možno, ale je to ten najlepší spôsob, ako získať prvé skúsenosti bez toho, že by ste sa báli, či výsledný program bude fungovať.

My pôjdeme podobnou cestou. Predložím vám jednoduchý, ale funkčný program, na jeho popise si vysvetlíme nielen príkazy TP, ale hlavne filozofiu programovania. Potom ho spoločne vylepšíme, čím si objasníme ďalšie a ďalšie príkazy, metódy a postupy. Od neho sa pokusíme odvodiť programy, ktoré budú vhodné na pravidelné používanie. Veď načo by boli programy iba cvičné, nepotrebné v bežnom živote?

Dosť už tohto beletristického štýlu, vhodného do románov, a začnime programovať. Ale to už nabudúce.

STRETNUTIE S PASCALOM

2. časť

Keď Niklaus Wirth z Vysokej školy technickej v Curychu vytvoril vyšší programovací jazyk a zvolil jeho názov na počesť francúzskeho filozofa, matematika a fyzika **Blaise Pascala** (1623 – 1662), ešte nevedel, akého enormného rozšírenia sa tento jazyk dočká. Preto aj my pristúpme k prvej lekcii.

Ešte niekoľko obecných zásad

● Každý programátorský problém sa dá riešiť viacerými spôsobmi. Preto nikdy nepovažujte nejaké riešenie za absolútne správne a jediné, aj keď sa to na prvý pohľad zdá. Na druhej strane by nebolo dobré snažiť sa napísať rutinu, ktorá vyrieši ten samý problém, inak len preto, aby to bola iba vaša vlastná interpretácia algoritmu. Zvolenie vhodného kompromisu vedie k veľmi slušným výsledkom.
● Nepodliehajte dezilúzii hneď po prvých neúspechoch, ale neočakávajte, že po niekoľkých lekciách napíšete aplikáciu o desaťtisícich riadkoch.
● Neočakávajte od tohoto seriálu kompletnú učebnicu programovania. To nie je z rozsahových dôvodov časopisu možné a preto niektoré veci sa pokúste naštudovať alebo vyriešiť sami, poprípade v blízkom okolí.

A niekoľko praktických zásad

● originály zdrojových textov, alebo funkčné verzie programu si VŽDY odzálohujte na bezpečné miesto a médium, najlepšie na disketu.
● Turbo Pascal (TP) odkladá aktuálny zdrojový text s príponou xxx.PAS a poslednú textovú zmenu v zdrojáku s príponou xxx.BAK. Avšak nepozná (ani to nie je možné) logické zmeny verzií daného programu. Preto je dobré uložiť logickú zmenu – úpravu pod iným menom, napr. s pridaním čísla na koniec mena (TELEFON2.PAS). Uvidíte, že sa budete musieť niekedy vrátiť aj o niekoľko variant dozadu, keď zistíte, že „tadiaľto cestička nevedie“.
● Nezabudnite na komentáre v programe! Po pol roku budete pozerať do listingu vášho programu a v zložitejších rutinách len s problémami pochopíte, čo ste sami vytvorili. (Vlastná skúsenosť.)
● Nebojte sa experimentov. Skúšajte, skúšajte, skúšajte!

Obecná štruktúra programu

v TP je na listingu č.1. Zvýrazneným slovám (a ešte mnohým ďalším) sa hovorí *rezervované*. **Nikdy** ich nesmieme použiť ako názvy procedúr alebo premenných. A vôbec, TP nás pri preklade na to upozorní sám.

Zdrojový text programu sa skladá s troch hlavných častí:
● ***hlavička programu*** – začína slovom program, kde nasleduje meno programu. Toto nie je povinné, ale veľmi vhodné. Tu patrí aj klauzula **uses**, ktorou určujeme, ktoré knižnice budeme používať.
● ***deklaračná časť*** – tu sa definujú hlavné objekty (pre skúsenejších – nezamieňať s objektami v OOP) jazyka. Deklarácia je informáciou pre prekladač jazyka o objektoch vytvorených užívateľom, ktoré sa budú vyskytovať v programe, a o ich vlastnostiach. Deklarácie sú povinné a každý objekt, ktorý je v programe použitý a nie je štandartný (t.j.poskytovaný definíciou jazyka), musí byť *pred použitím deklarovaný*. Na základe informácií získaných v deklaráciách môže prekladač prevádzať kontrolu správneho použitia objektov a odhaliť mnohé chyby. To zrýchľuje a zefektívňuje ladenie. Práca naviac so zápisom deklarácií sa vždy vyplatí!
● ***príkazová časť*** – tu je popísaný algoritmus programu, ktorý sa skladá z príkazov. Príkazom môže byť aj nami vytvorená funkcia alebo procedúra, definovaná v deklaračnej časti.

V zdrojovom texte sa nachádzajú už vyššie spomínané poznámky. Tie je možné zadávať dvoma spôsobmi, a to v zložených zátvorkách, napr.

{ toto je poznámka }

alebo v jednoduchých zátvorkách s hviezdičkou, napr.

(* aj toto je poznámka *).

```
program Meno_programu;

uses meno_knižnice_1, meno_knižnice_2,...meno_knižnice_X ;

(*************** Deklaračná časť *****************)

const                (konštanty)

type                 (typy)

var                  (premenné)

function              (funkcie)
begin
  tu je telo funkcie
end;

procedure             (procedúry)
begin
  tu je telo procedúry
end;

(************** Príkazová časť ******************)

BEGIN

  tu sa nachádza popis vlastného tela programu, ktoré
  sa skladá z príkazov, funkcií alebo procedúr.

END.
```

Musíme dbať na párny počet zátvoriek, lebo by mohol prekladač hlásiť chyby, ktoré by neboli na prvý pohľad jasné.

Najdôležitejšie pri písaní zdrojových textov je dodržiavanie ukončovania riadkov. Najčastejším znakom je bodkočiarka(";"). Spočiatku to môže robiť problémy, ale po čase sa vám to vžije do krvi.

Začíname

Po spustení TP sa ocitneme v modrom vývojovom prostredí IDE (Integrated Development Environment = Integrované vývojové prostredie). Nesmieme zabudnúť správne nastaviť adresáre vo voľbe Option@Directories, aby sa nám zdrojové texty, unity a preložené programy netúlali po disku medzi systémovými súbormi. Ostatné voľby nastaví TP pri inštalácii štandartne a nie je ich spravidla potrebné meniť. Keď sa tak stane, upozorníme na to.

V IDE sa programy nielen píšu, ale aj odlaďujú. TP totiž dokáže odlaďovaný program pri výskyte chyby pozastaviť, ukáže na miesto kde chyba vznikla, chybu popíše a pomocou helpu napomôže pri jej odstránení. Pod ladením programu rozumieme postupné skúšanie celku alebo určitých súčastí programu, ošetrenie syntaktických alebo logických chýb tak, aby program robil to, čo má. Na to slúžia rôzne spôsoby, ktoré si postupne preberieme.

A tak môžeme tvoriť náš

Prvý program

v Turbo Pascale. Čo bude jeho úlohou? Bude to softwarový telefónny zoznam priateľov a známych, ktorý umožní zaradenie jednotlivých osôb a ich prezeranie. Na začiatok ako základ to stačí. Ďalšie príjemné funkcie, ako listovanie, prehľadávanie, vytlačenie, automatické vytáčanie čísel a pod. dopracujeme postupne. Program sa nazýva TELEFON a jeho výpis je na listingu č. 2.

Je to veľmi jednoduchý program, ktorý sa v tejto verzii užívateľovi javí takto:

Po spustení už preloženého programu sa obrazovka vymaže a objaví sa hláška „Zadaj meno“. Program očakáva zadanie znakov z klávesnice. Po napísaní mena a stlačení ENTER sa objaví druhá hláška „Zadaj priezvisko“ a očakáva sa znovu vstup

z klávesnice. Meno aj priezvisko môžu byť ľubovolné znaky, symboly a čísla. Ak by boli dlhšie ako 15 znakov, program príme iba prvých 15 znakov, ostatné sa odrežú. Po stlačení Enter sa objaví tretia hláška „Zadaj číslo“. Tu sa očakáva vstup číslic. Pri vložení iných znakov program skončí s chybou. Toto sa opakuje dovtedy, pokým pri hláške „Zadaj číslo“ nestlačíme nulu a Enter. Vtedy program ukončí zadávanie osôb a na obrazovke vypíše všetky zapísané osoby s príslušnými údajmi. Po následnom stlačení Enter program skončí. Vytvorí však na disku, odkiaľ bol spustený, súbor s DOS-ovským menom TELEFON.ZOZ, kde sú uložené údaje, ktoré sme zadali z klávesnice.

Teraz sa pozrime na program s pohľadu programátora. Na listingu č. 2 si všimnime, že sú vyznačené vyššie spomínané časti štruktúry každého programu. Na pravej strane v zložených zátvorkách sú u väčšiny riadkov príslušné poznámky, ktoré popisujú úlohu jednotlivých príkazov. Komentáre sú pomerne podrobné kvôli úplným začiatočníkom. Skúsenejší programátori nemusia písať veci, ktoré sú im známe.

Prejdime si spoločne jednotlivé riadky podrobnejšie, ako je komentár v zátvorkách.

Pod menom programu klauzulou **uses** je priradená knižnica CRT, ktorá je súčasťou balíku TP. Zbytočne by ste na disku hľadali súbor CRT.TPU. TP má totiž svoje základné knižnice uložené v súbore TURBO.TPL (Turbo Pascal Library). V tejto knižnici sa nachádza definícia príkazu Clrscr, ktorý maže obrazovku počítača.

V deklaračnej časti je nami nadefinovaný nový typ osoba, ktorý je typu record (záznam) a skladá sa z troch položiek. Meno a priezvisko sú typu string (reťazec), takže môžu prijímať všetky dostupné znaky, symboly a číslice. Tretia položka cislo je typu **integer**, čo značí celé čísla od -32768 do 32767. Pri zadaní iného znaku vznikne chyba! Číslo [15] u položiek meno a priezvisko určuje počet možných znakov, teda 15. Môžeme si zvoliť aj inú hodnotu.

Pod slovíčkom **var** deklarujeme premenné (variables). Prvá sa nazýva priatel a je typu osoba, ktorý sme deklarovali vyššie. Bude prezentovať údaje o našich priateľoch a známych. Druhá, *zoznam* je typu **file of** osoba, čo značí, že je to „súbor osôb“. Posledná premenná kolko je už spomínaného typu integer a prezentuje počítadlo, ktorým budeme počítať počet záznamov (osôb) v súbore. Toľko deklaračná časť.

Príkazová časť sa nachádza medzi slovami **BEGIN** a **END.** s bodkou. Prvý príkaz tejto časti Assign priraďuje meno externého súboru (napr. na disku) premennej typu súbor (file), v tomto prípade premennej zoznam. Naďalej budeme pracovať už len s premennou zoznam.

Druhý príkaz Rewrite vytvára súbor s priradeným menom na disku. Ak už na disku existuje súbor toho istého mena, zruší ho a vytvorí nový. Súbor na disku sa bude nazývať TELEFON.ZOZ. Názov súboru záleží len na vôli programátora.

V dvojici slov **repeat – until** (opakuj – pokým) je skupina príkazov na výpis hlásenia na obrazovku (*write*) a čítanie znakov z klávesnice (*readln*). Všimnime si spôsob prístupu k položkám meno, priezvisko a cislo cez premennú priatel. Hovoríme tomu bodkový prístup. Môžeme povedať, že príkazom Readln sa načítajú znaky z klávesnice a naplnia sa položky meno, priezvisko a cislo premennej priatel, ktorá je typu osoba. Možno sa to zdá zložité, ale je potrebné si toto objasniť.

Ak premenná priatel.cislo bude rôzna od nuly, tak sa obsah celej premennej priatel (teda všetkých jej troch položiek) uloží do premennej zoznam, čo nie je nič iné, len súbor na disku s menom TELEFON.ZOZ. Tento cyklus sa opakuje dovtedy (until), pokiaľ nie je splnená podmienka, že priatel.cislo sa rovná nule.

Príkaz Close uzavrie súbor na disku. Tento príkaz je pomerne dôležitý, lebo kedy sme ho opomenuli, môžu sa stratiť niektoré údaje! Princíp práce so súbormi si vysvetlíme neskôr.

Dostávane sa k druhej časti programu.

Začína zmazaním obrazovky. Príkaz Reset je ďalší z príkazov pre prácu so súbormi. Otvára existujúci súbor na disku. (Nech nás nemýli zľudovelý počítačový význam reset počítača = vymazanie!). Nastavíme počítadlo – premennú kolko na nulu. Pomocné premenné musíme pred použitím nastaviť na určitú hodnotu, inak by sme dostávali nezmyselné výsledky a zakaždým iné. Prichádza už známy cyklus **repeat – until**, tentokrát však sa príkazom read prečíta so súboru (prezentovaným premennou zoznam) do premennej priatel obsah údajov o osobe. Zväčšíme počítadlo o jeden. Prebehne vypisovanie na obrazovku, najprv hlášky, uvedené v apostrofoch, potom obsahy jednotlivých položiek meno, priezvisko a cislo premennej priatel. Príkaz Writeln vypíše prázdny riadok. Cyklus končí, keď sa zistí koniec súboru (Eof=end of file). Vypíše sa hláška o počte záznamov a samotný počet záznamov, prezentovaný premennou kolko. Procedúra Readln očakáva stlačenie klávesy Enter. Známym príkazom Close nesmieme zabudnúť uzavrieť diskový súbor.

Sme na konci nášho prvého programu. Teraz ho prepíšeme v TP. Dajte pozor pri prepisovaní na správnosť rôznych znamienok, čiarok a bodiek. TP je veľmi prísny a žiadnu inú variantu nepripustí!

Asi ste si všimli „rebríkovitú“ úpravu zdrojového textu. Táto je pre Turbo Pascal typická a pri viacriadkových textoch je doslova potrebná. Zvyknite si na ňu, už ju nikdy nebudete chcieť opustiť. Po napísaní zdrojového textu pristúpime k samotnému ladeniu programu. Ale to nabudúce.

STRETNUTIE S PASCALOM

3. časť

Určite to poznáte! Prvý zápal z počítača, kedy spúšťate jednu hru za druhou, je zápal očných spojiviek od prebdených nocí nad novozískaným „členom rodiny“. Niektorým sa to tak zapáči, že ostávajú pri hrách už navždy, a tak sa rodia špecialisti na DOOMy a ostatné chuťovky tejto novodobej drogy. Zvyšok prechádza druhou etapou vývoja. Skúšajú rôzne textové a grafické editory, kalkulačné a databázové programy alebo celkom užitočné „kusovky“, ktoré im nakoniec priľnú k srdcu, lebo ich pomerne úspešne používajú pri svojej profesii. (Aj tu sa nájdu rôzni zberatelia, ktorí sú schopní vymenovač desiatky programov toho istého typu, všetky pracne držia na harddisku, ale aj tak píšu nakoniec v tešesťstodvojke...). Drvivá väčšina ľudí z určitých pochopiteľných dôvodov končí na tejto platforme vývoja počítačníkov, pretože im tento stupeň počítačovej gramotnosti stačí. Hovorí sa im užívatelia. Ale z tejto skupiny sa vyčlenia nadšenci, ktorých lákajú počítačové diaľavy. Orientujú sa spravidla iba na jednu úzko vymedzenú oblasť kompjúterového spektra, kde dosahujú vysoko profesionálnych výsledkov. Rôzni grafisti, CADisti, bejzisti (pôvodne som chcel napísať basisti, ale z určitých dôvodov som radšej použil fonetický tvar... aby sa niekto neurazil) a hlavne sieťari sú Páni a Dámy (áno aj Dámy) s veľkým P a D. Bez ich nadšenia a profesionality by neexistovali rôzne efektné reklamy, časopisy, designy strojov (alebo aj šiat) a ani najrozmáhajúcejšia vetva ľudskej elektronickej komunikácie – sietí. Ľudia sú tvory bádavé, a tak časť prejde treťou etapou vývoja. Nestačí im používanie toho-ktorého produktu, chcú sami vytvoriť svoj vlastný program, ktorý si „ušijú“ presne na mieru. Vstupujú na horúcu pôdu programátorskú, nevediac, že je to oblasť plná horkosti a sklamaní, ktoré nakoniec absolútne preváži nepopísatelná radosť z fungujúceho programu. Keď vytvoria produkt, ktorý začne používať určitá skupina užívateľov, získavajú nielen hmotný zisk, ale aj renomé. Vedomie, že užívatelia používajú produkty programátorov, je asi najväčšou hybnou silou, ktorá akoby Švihnutím čarovného prútika zastavuje bolesti hlavy kvôli prebdeným nociam nad listingami programov, dáva silu do boja s rodinným rozpočtom alebo drobnou nevraživosťou rodiny pre čas, ktorý ste predtým venovali jej. Lebo verte, milé dámy a páni, že o peniaze a čas vás pripraví

žena najpríjemnejšie, alkohol najbolestnejšie a počítač najrýchlejšie. Stač sa programátorom nie je otázka rozhodnutia. Je to vec, ktorá chce trochu talentu, chuti a Štúdia. Zatiaľ čo prvé dve vlastnosti sú danosti a prejavia sa okamžite, Štúdium vyžaduje čas. So Štúdiom súvisí výber prostriedkov, ktorými chceme dosiahnuť svoje programové predstavy a ciele. Ja som si z určitých dôvodov (schválne neuvádzam akých, lebo nechcem rozvíriť nikdy nekončiacu polemiku, či Pascal alebo C) vybral programovací jazyk Turbo Pascal od firmy Borland. Jazyk vhodný ako na výuku programovania, veď bol k tomu zrodený, tak aj na tvorbu veľmi solídnych projektov.

Čo je pascal

Pascal je vyšší programovací jazyk. Vytvoril ho pán Wirth ako pomôcku programovania pre Študentov. Vďaka jednoduchému osvojeniu a veľkej modularite sa veľmi rýchlo rozšíril a v určitých implementáciách existuje dodnes a dni jeho zániku aj napriek rôznym predpovediam sú stále v nedohľadno. To, že je vyšším programovacím jazykom, znamená, že odstraňuje problémy strojovo orientovaných jazykov, kde program bola postupnosť elementárnych inštrukcii bez kontroly správnosti zápisu a činnosti. Vyšší programovací jazyk zavádza vhodné štruktúry, ktoré zvyšujú zrozumiteľnosť napísaných programov, a tým zjednodušujú programovanie. Medzi prvé aj u nás veľmi známe implementácie jazyka pascal patrí program od firmy HISOFT, ktorý fungoval na starom dobrom Sinclairi. Medzi pécečkármi je snáď najznámejšia implementácia Wirthovho pascalu od firmy BORLAND International, ktorej vlastníkom je Phil Kahn. Volá sa TURBO PASCAL a jeho zatiaľ posledná verzia nesie číslo 7. Najobľúbenejšia bola verzia šesť, a keďže sa sedmička v základoch od šestky nelíši a je dostupná na mnohých počítačoch či už v školách alebo doma, budeme sa venovať predposlednej verzii. Musíme si uvedomiť, že TURBO PASCAL (ďalej už len P) je kompiler. To znamená, že prekladač preloží zdrojový text do spustitelného súboru s príponou .EXE, ku ktorého činnosti už nie je potrebný samotný TP. Teda doma v TP napíšem program, po preložení ho prenesiem na počítač, kde žiadny pascal nie je a môžem ho spustiť. Prostrediu samotného TP hovoríme vývojové prostredie.

Čo je Turbo Pascal 6.0

TP 6.0 je programový integrovaný balík, software, ktorý v komplexe obsahuje vývojové prostredie, prekladač, knižnicu podprogramov, rôzne utility, help, vzorové príklady a dokumentáciu (v angličtine).

Dodáva(l) sa v dvoch variantoch, a to ako Turbo Pascal 6.0 a Turbo Pascal 6.0 Professional. Niekde na našom počítačovom trhu bol aj český help, ktorý nahradil anglickú verziu.

Čo budeme potrebovať

TP 6.0 pobeží na ľubovoľnom PC s pamäťou aspoň 1 MB a miestom na disku do 7 MB. (Ja som začínal s TP 5.5 na PP 06 bez harddisku a s 2 mechanikami a CGA kartou. To boli časy... prekladu.). Najvhodnejšia je VGA karta a farebný monitor, ale postačí aj EGA alebo monochrom. Ďalej myš, ktorá nie je nevyhnutná, ale veľmi spríjemní prácu v prostredí TP. Ak máte tlačiareň alebo aspoň prístup k nej, ušetríte si časté omyly. Výpis z tlačiarne sa študuje podstatne lepšie ako výpis na obrazovke, ktorá vám poskytuje stránku o 25 riadkoch. Ale ani tlačiareň nie je nevyhnutná. Čo je nevyhnutné, je samotný TP 6.0. (Hmm, tu musím upozorniť na legálnosť programu, hmm, hmm...). Tí, ktorí majú TP 7, môžu pracovať s nami, odlišnosti nebudú. (Zatiaľ). Veľmi by sa hodil aj vyššie spomínaný český help, nebol ani drahý, za pár stoviek sa dá získať a stojí to za to. Pre tých, čo sú dobrí v angličtine, to neplatí.

Ako sa učiť programovať

Teórií o výuke programovania sa už napísalo dosť. Každá je v niečom výhodnejšia od druhých, každý vyučujúci si ju upraví podľa svojej predstavy. Metóda, keď sa preberá príkaz TP abecedne jeden za druhým, je asi tá najmenej šťastná. Začínal som tak, že som dostal zdrojový text hotového fungujúceho programu. Ten som si po preložení vyskúšal, zistil som, čo vlastne robí. Potom som program krokoval a sledoval jeho činnosť. V dostupnej literatúre som naštudoval konkrétny príkaz jednotlivého kroku, poprípade zmenil jeho parametre. To viete, že som spravil aj také zásahy, kedy už program nefungoval vôbec. Ale vďaka odloženému originálu a posledným zmenám som sa mohol vrátiť o jeden-dva kroky späť a skúšať znovu.

Zdá sa vám to ťažkopádné? Možno, ale je to ten najlepší spôsob, ako získať prvé skúsenosti bez toho, že by ste sa báli, či výsledný program bude fungovať. My pôjdeme podobnou cestou. Predložím vám jednoduchý, ale funkčný program, na jeho popise si vysvetlíme nielen príkazy TP, ale hlavne filozofiu programovania. Potom ho spoločne vylepšíme, čím si objasníme ďalšie a ďalšie príkazy, metódy a postupy. Od neho sa pokúsime odvodiť programy, ktoré budú vhodné na pravidelné používanie. Veď načo by boli programy iba cvičné, nepotrebné v bežnom živote?

Dosť už tohto beletristického štýlu, vhodného do románov, a začnime programovať. Ale to už nabudúce.

STRETNUTIE S PASCALOM

4. časť

Opakovanie

V predchádzajúcej časti sme si povedali niečo o chybách programu, o tom, ako sa prejavujú a naznačili postup pri odstraňovaní. Objasnili sme si najzákladnejšie postupy pri ladení programu, jeho spustení a preklade. Vysvetlili sme si činnosť a zmeny v programe TELEFON2.PAS. A ako dopadla

Domáca úloha?

Isto ste si všimli, že otázky z cvičenia neboli iba teoretické, ale popisovali neštandartné správanie sa nášho programu v určitých situáciách. Ako v správnej škole si úlohu objasníme.

Prečo sa program správa neštandartne?

Sami viete, že každý program má drobné vady. Aj keď základný algoritmus pracuje bezchybne, nadstavba nad ním (interface, čítaj interfejs alebo aj užívateľské prostredie), zabezpečujúca jeho komunikáciu s užívateľom a ostaným okolím má určité zádrhely. Keď začnete programovať vlastné rozsiahlé aplikácie zistíte, že 10% času zaberie tvorba algoritmu a procedúr, a 90% tvorba užívateľského prostredia.

Úlohou užívateľského prostredia je eliminovať všetky možné nesprávne postupy zo strany užívateľa vytvoreného programu tak, aby nedošlo k žiadnej nežiadúcej situácii alebo dokonca zrúteniu sytému. My vieme, že v určitej premennej je nutné zadávať iba numerické hodnoty, ale čo ak tam užívateľ vloží napr. písmeno? Program havaruje. A preto je nutné všetky tieto chyby vychytať. Je to práca skutočne náročná a zapamätajme si, že autor programu nie je objektívny, ak sa jedná o kontrolu programu z pohľadu užívateľa. Preto prední výrobcovia software vytvárajú tzv. beta verzie, ktoré rozdávajú zdarma k testom mnohým užívateľom. Tí otestujú produkt vo (skoro) všetkých možných situáciách, zisťujú chyby a problémy, ktoré potom autori pracne odstraňujú. A že to trvá niekedy veľmi dlho, dokazujú napr. aj časové posuny v niekoľkokrát ohlasovanom uvedení Windows 95.

Ešte predtým, ako pristúpime k vysvetleniu precvičovacích otázok, mali by sme si povedať niečo o súboroch. Súbory sú veľmi dôležitou súčasťou mnohých aplikácií, a preto správne s nimi pracovať patrí medzi pevné základy programovania.

Súbory a práca s nimi

Súbory sú chápané ako premenné typu *file*. Sú tri triedy pascalovských súborov: *typové, textové a netypové.* Syntaxiou sa budeme zaoberať postupne.

Skôr ako začneme premennú súbor využívať, musíme ju spojiť s fyzickým súborom procedúrou *Assign.* Ako typické fyzické súbory menujme diskové súbory, ale môžu nimi byť aj zariadenia, napr. klávesnica alebo obrazovka. Informácie sa do súboru ukladajú zápisom, získavajú sa čítaním. Aby mohla táto komunikácia s fyzickými súbormi úspešne prebiehať, musí byť súborová premenná „otvorená". Nový súbor sa zakladá procedúrou *Rewrite*, už existujúci môže byť otvorený procedúrou *Reset*. **Textové** súbory otvorené *Resetom* sú prístupné iba pre **čítanie**, otvorené pomocou *Rewrite* alebo *Append* iba pre **zápis. Typové** a **netypové** súbory umožňujú obidva smery prístupu, **čítanie** aj **zápis**.

Súbor je chápaný ako postupnosť *položiek*, z nich každá je typu predpísaného pre komponentu (alebo ak chceme – vetu) daného súboru. Každá položka má svoje *poradové číslo* charakterizujúce polohu v súbore. Prvá položka súboru má číslo 0 (nula). Súbory sú zvyčajne prístupné sekvenčne, t.j. položky sú čítané alebo zapisované tak, že *ukazateľ pozície v súbore* sa po každej operácii automaticky nastavuje na následujúcu hodnotu. K určeniu aktuálnej pozície ukazateľa (niekedy nazývaného i smerník) v súbore slúži funkcia *FilePos*. Aktuálnu veľkosť súboru zistí funkcia *FileSize*. Keď program ukončuje spracovávanie súboru, je nutné súbor korektne uzavrieť procedúrou *Close*, inak by mohlo dôjsť k strate dát, ktoré boli ešte vo vyrovnávacej pamäti. Volanie štandartných I/O operácií nad súbormi má automatické riadenie chybových stavov. Ak sa vyskytla chyba, program je ukončený so zobrazením chybovej správy. Toto automatické riadenie chýb môže byť *povolené* alebo *zakázané* direktivou kompilátoru {$I+} a {$I-}. Pokiaľ je obsluha I/O chýb vypnutá, nevedie chyba k abnormálnemu ukončeniu programu, ale ju môžeme programovo ošetriť. Informácia o výsledku I/O operácii je prístupná volaním funkcie *IOResult*.

Štandartné podprogramy pre všetky typy súboru

Uveďme iba ich význam:

Procedúry:

Assign	priradí k premennej súbor meno fyzického súboru
ChDir	zmena aktuálneho adresára
Close	uzavrenie otvoreného súboru
Erase	zmazanie fyzického súboru
GetDir	zistí aktuálny adresár
MkDir	tvorba podadresára
Rename	premenovanie súboru
Reset	otvorenie existujúceho súboru
Rewrite	vytvorenie nového súboru a jeho následné otvorenie
RmDir	zrušenie prázdneho adresára

Funkcie:

EoF	indikácia konca súboru
IOResult	status predchádzajúcej I/O operácie

Štandartné podprogramy pre textové a typové súbory

Musíme si uvedomiť, že typ *Text* sa líši od typu *file of char*.

Otvorený textový súbor je chápaný zvláštnym spôsobom: Jeho obsahom je totiž postupnosť znakov rozdelených do riadkov ukončených znakom konca riadku *carriage-return* (CR), ktorý môže byť nasledovaný znakom *line-feed* (LF).

Procedúry:

Append	otvorenie existujúceho súboru pre jeho doplnenie
Flush	zápis obsahu bufferov do výstupných súborov
Read	čítanie jednej alebo viac hodnôt zo súboru do jednej alebo viacerých premenných (text) načítanie hodnoty zo súboru (typové)
ReadLn	vykonáva to samé ako Read a naviac skok na začiatok ďalšieho riadku súboru (iba pre text)
SetTextBuf	priradí textovému súboru I/O buffer (vyrovn. pamäť)
Write	zápis jednej alebo viac hodnôt do súboru (text), zápis hodnoty do súboru (typ)
Writeln	to samé ako Write a naviac zapíše znak konca riadku (len pre text)

Funkcie: (len text)

Eoln	indikácia konca riadku
SeekEof	nastavenie pozície znaku a test konca súboru
SeekEoln	nastavenie pozície znaku a test konca riadku

Štandartné podprogramy pre netypové súbory

Netypové súbory využívajú nižšiu úroveň I/O operácií, ktoré umožňujú priamy prístup na disk nezávisle na type alebo štruktúre dát. Deklarujú sa slovom *file* (a nič viac).

S výnimkou procedúr Read a Write využijeme všetky podprogramy známe z typových súborov. Miesto nich sú definované procedúry s vysokou rýchlosťou datového presunu:

BlockRead	načíta do premennej jednu alebo viac viet
BlockWrite	zapisuje jednu alebo viac viet z premennej do súboru

Následujúce operácie je možné použiť pre ľubovoľný typ súboru s výnimkou textového:

FilePos	zisťuje aktuálnu pozíciu smerníka v súbore
FileSize	vracia veľkosť súboru
Seek	nastaví smerník na posíciu v súbore
Truncate	skráti súbor po aktuálnej pozícii

Toľko veľmi potrebná teória pre prácu so súbormi.

Vráťme sa teraz k precvičovacím otázkam:

1/ Ak zadáme číslo väčšie ako 4, nič sa nedeje. Väzba *repeat readln(co) until* v príkazovej časti totiž očakáva iba vstup čísel 1,2,3 a 4. Kritická situácia nastane, ak zadáme miesto ľubovoľného čísla nejaký znak. Program končí s chybou 106, čo je vadný numerický formát. Odstránenie tejto chyby je pomerne jednoduché, ako je vidieť na listingu TELEFON3.PAS. Premennú *co* zmeníme na typ char – znak, a príkazom *readkey* testujeme stlačenie ľubovoľnej klávesy. Hodnota klávesy sa uloží do premennej *co*. Riadok *until co in ['1','2','3','4']* kontroluje hodnotu *co* a väzba *repeat – until* neskončí, pokým nie je jedna z vymenovaných (1 alebo 2 alebo 3 alebo 4) kláves stlačená. Vyskúšajte, že už nie je potrebné po voľbe položky menu stláčať Enter.

2/ Ak pri zadávaní telefonného čísla zadáme nenumerickú hodnotu, nastáva ten istý problém ako v bode 1/. Príkaz *Readln (priatel.cislo)* očakáva iba číselný formát. Odstránenie takejto chyby vyžaduje zmenu štruktúry súboru. Položku *cislo* záznamu *osoba* zmeníme na typ reťazec-string (pozri listing TELEFON3.PAS). S tým súvisí zmena kontroly, keď chceme ukončiť zadávanie dát. Nebudeme kontrolovať nulu, ale reťazec „koniec". To však značí, že nemôžeme použiť datové súbory vytvorené verziou TELEFON2. Musíme naplniť súbory s novou štruktúrou. Z tohto dôvodu je táto oprava závažná.

3/ Ak má súbor viac záznamov ako je voľných riadkov na obrazovke (v našom prípade viac ako 20, teda 21 a viac) dochádza v príkaze *GotoXY* k pretečeniu parametra Y. Pozrime sa bližšie na procedúru *Vypis.* Keď procedúra načíta napr. piaty záznam zo súboru, v príkaze *GotoXY* nastane tento stav: X = 1,(18,35), Y = kolko + 5 = 5+5 = 10. Kurzor sa postaví príkazom *GotoXY(1,10)* na prvý stĺpec a desiaty riadok v súradniciach obrazovky, kde sa príkazom *Write* vypíše položka *meno* piate-

ho záznamu. Čo nastane, ak sa má vypísať 21. záznam? X=1, Y= kolko +5 = 21+5 = 26. Príkaz *GotoXY(1,26)* sa neprevedie a výpis nie je korektný, lebo obrazovka má iba 25 riadkov.

Odstránenie tejto chyby vyžaduje úpravu procedúry *Vypis*: TP vie používať pre výpisy na obrazovku tzv. okná (Window). Nedajte sa mýliť, podobnosť so software firmy Microsoft je v tomto prípade iba fonetická. Obrazovka počítača je z programového hľadiska jedno veľké okno, ktorého ľavý horný roh má súradnice 1,1 a pravý dolný roh má súradnice 80,25. Implicitne je teda okno obrazovky definované ako *Window(1,1,80,25)* a má 80 stĺpcov a 25 riadkov. Okná môžu byť v TP definované v ľubovolnom mieste obrazovky a v ľubovolnej veľkosti. V takto vytvorenom okne môžeme použiť všetky dostupné procedúry a funkie, platné pre obrazovku. Musíme si uvedomiť, že ľavý horný roh novovytvoreného okna bude mať pre procedúry a funkcie súradnice 1,1! Napr. ak nadefinujeme okno príkazom *Window (1,6,80,25)*, ľavý roh začne na prvom stĺpci a šiestom riadku obrazovky. Následná procedúra *GotoXY(1,1)* postaví kurzor na pozíciu prvého riadku a prvého stĺpca, ale v novom okne, teda v skutočnosti na prvý stĺpec a šiesty riadok fyzickej obrazovky monitoru. Zložité, však. Ale uvídíte, ako často to budete využívať. Ale vráťme sa k procedúre *Vypis*. *Window(1,6,80,25)* nastaví nové okno pre výpis na obrazovku. Prečo až na šiesty riadok? Lebo prvých päť obsadí procedúra *Hlavicka*. Nahradíme premennú *kolko* premennou *pos*, ktorá bude zabezpečovať nastavenie súradnice Y. V novom okne už nemusíme pripočítavať +5. Po výpise položiek záznamu príkazom *if pos mod 19 = 0* testujeme, či sa vypísalo 19 riadkov na obrazovke. Ak áno, vynulujeme *pos* a na 20.riadok vypíšeme hlásenie. Na príkaze *readkey* sa program pozastaví a očakáva vstup z klávesnice. Po stlačení sa do premennej *ch* uloží ASCII hodnota stlačeného znaku. Ak sa táto hodnota rovná "a" (97) alebo "A" (65), tak sa okno vymaže a výpis pokračuje od prvého riadku tohto okna nasledujúcim záznamom. Ak stlačíme inú klavesu, okno sa vymaže, nastaví sa štandartná veľkosť obrazovky a *Exit* ukončí vykonanie procedúry *Vypis*. Nesmieme zabudnúť na nastavenie štandartnej obrazovky (1,1,80,25) na konci procedúry. Bližší popis je v komentároch listingu TELEFON3.PAS.

Keď podrobíme program TELEFON2 poriadnym testom, zistíme, že ak nemáme ešte žiaden datový súbor TELEFON.ZOZ a vybrali sme voľbu 2/ alebo 3/ z menu, program skončí s chybou 002 (nemôže nájsť súbor). Preto si v TELEFON3 deklarujeme novú funkciu *Nie_je_subor*, ktorá je typu boolean. To znamená, že môže nadobúdať jednu z dvoch hodnôt – funkcia je pravdivá (true) alebo nie je pravdivá (false). Použijeme vyššie spomínanú funkciu *IOResult*, ktorá zistí, či otvorenie *(Reset)* súboru prebehlo korektne (hodnota 0) alebo nie. Ak súbor neexistuje (IOResult je rôzne od nuly), vypíše sa hlásenie a funkcia *Nie_je_subor* je pravdivá. V opačnom prípade získa hodnotu false (nie je pravda, že nie je súbor – negácia negácie). Túto funkciu vložíme do procedúr *Pridaj* a *Vypis*. Ak súbor nebude existovať, funcia *Nie_je_subor* nadobudne hodnotu true a *Exit* ukončí činnosť procedúr.

Pre odskúšanie činnosti programu TELEFON3 použijte krokovanie F7 a F8. Mnohé veci sa vám ujasnia.

Precvičovacie otázky

1/ Čo znamená parameter ^*g* v procedúre Write ? Odskúšajte!

2/ Môžeme na riadku *if Nie_je_subor =true then* ... vynechať "=true"? A prečo?

3/ Ako zabezpečíme, aby sa súbor TELEFON.ZOZ nachádzal v inom adresári ako hlavný program ?

Naša kartotéka telefónnych čísel priateľov má ešte stále chybičky. Ale nemá to hlavné, čo katrotéku robí kartotékou – výpis na tlačiareň a to najhlavnejšie – vyhľadávanie. Ale o tom nabudúce.

STRETNUTIE S PASCALOM

5. časť

Opakovanie

Minule sme si povedali niečo o súboroch a práci s nimi. Nepodceňujte to, program ako taký sa skladá z jedného súboru len málokedy. Prezrite si ľubovoľný program na vašom disku. Presvedčíte sa, že väčšina z nich sa skladá z viacerých väčších či menších súborov, ktoré slúžia napr. na konfiguráciu programu (spravidla s príponou .cgf), na odkladanie dát, rôznych výstupov, grafov, hlásení-reportov a pod. Tieto navzájom korelujú (nemýliť s Corel!-om!) a tak vytvárajú určitý „balík“ súborov, ktorý môžeme nazývať aj projekt. Typickým a najmarkantnejším príkladom sú databázové a tabuľkové programy.

Zároveň sme sa pokúsili urobiť náš program trochu „user-friendly“, teda sme odstránili podstatné vady programu. Píšem „pokúsili“, lebo ako hovorí jeden z Murphyho zákonov „...odstránením jednej chyby v programe sa vnesie chyba nová“. Ale nový rok – nenový rok, treba skontrolovať

Domáce úlohy

1/ Parameter ^g nie je nič iné, ako iný zápis znaku 7 z tabuľky ASCII. Je to symbol bell – zvonec, archaizmus z dôb terminálov. Príkaz *write(^g)* spôsobí pípnutie reproduktora počítača.

2/ Funkcia *Nie_je_subor* je typu boolean, teda logická. Kontrola jej hodnoty môže byť zapísaná porovnávaním, napr.:

Nie_je_subor = true alebo

Nie_je_subor = false

alebo nastavením, napr.:

Nie_je_subor, kedy výraz považujeme za pravdivý (true), alebo *not(Nie_je_subor)*, kedy ho považujeme za nepravdivý (false).

3/ Odpoveď je veľmi jednoduchá. Ak ste pozorne preštudovali predchádzajúcu časť seriálu, viete, že na prácu so súborom v inom adresári použijeme procedúry týkajúce sa adresárov, teda MkDir, ChDir, GetDir a pod. Vzorové príklady nájdete v helpe TP.

Vráťme sa k nášmu programu. Vieme založiť novú databázu, vieme ju doplniť a vieme jej obsah vypísať na obrazovku. Chýba nám veľmi doležitá časť každého databázového programu, a tým je

Vyhľadávanie

Že neviete, na čo je to dobré? Predstavte si situáciu z reálneho života: Vezmite čo najhrubší telefónny zoznam (najlepšie New Yorku) a vyhľadajte, aké telefónne číslo má pán James C. Douglas, bývajúci na 5.Avenue! Koľko to asi bude trvať? Pri maximálnej lenivosti 3 minúty. To je slušné. Ale teraz inak! Na lístočku máte zapísané telefónne číslo, napr. 487-96-4582. Vyhľadajte v zozname, komu to číslo patrí! Ako dlho vám to bude trvať? Možno celé dni, nehovoriac o tom, že ho môžete spokojne prehliadnúť! Kvôli tej nekonečnej otročine sa stáva elektronické vyhľadávanie absolútnym víťazom. Preto aj my zamontujeme vyhľadávanie do nášho programu.

Algoritmus procedúry vyhľadávania

Mali by sme mať možnosť vyhľadávať podľa priezviska, ale aj podľa telefónneho čísla. Ak chceme vyhľadávať podľa priezviska, program by sa mal spýtať, čo a kde chceme hľadať. Tento reťazec hľadaných znakov zadáme s klávesnice. Procedúra otvorí datový súbor TELEFON.ZOZ, načíta prvú vetu súboru a v položke *priezvisko* skontroluje, či sa v nej nenachádza hľadaný reťazec znakov. Ak áno, vypíše na obrazovku všetky položky tejto vety, teda *meno*, *priezvisko* a *cislo*. Potom načíta druhú vetu súboru, opakuje kontrolu, načíta ďalšiu vetu a tak ďalej až do konca súboru. Ak sa v súbore v danej položke vety nachádza hľadaný reťazec, na obrazovke sa vypíše obsah všetkých položiek. Procedúra by mala zabezpečiť výpis na obrazovku, ak by bol počet nájdených viet väčší ako plocha obrazovky. Toto

sme riešili minule, teraz to len efektne použijeme. Ak je vyhľadávanie ukončené, uzavrieme súbor.

Obdobne to bude s vyhľadávaním podľa telefónneho čísla.

Pozrime sa bližšie na procedúru *Vyhladaj(podla_coho: string);* v listingu č.1. Parameter procedúry *podla_coho* je vstupný a určuje, či budeme vyhľadávať podľa priezviska alebo podľa mena. Všimnime si, že sme takto použili jednu procedúru dvakrát pre dve rozdielne úlohy. Pre akú úlohu bude použitá, určuje práve jej parameter. Takejto procedúre v TP hovoríme **parametrická**. Keby toto TP neumožňoval, museli by sme danú procedúru napísať pre jednotlivé úlohy zvlášť. Samotná procedúra je veľmi podobná procedúre *Vypis*. Skutočné vyhľadávanie zabezpečuje funkcia *pos(hladane,kde). hladane* je hľadaný reťazec, načítaný z klávesnice, *kde* určuje položku, v ktorej sa bude vyhľadávať – *priatel.priezvisko* alebo *priatel.cislo*. Ak sa reťazec *hladane* nachádza v položke vety *kde*, funkcia vráti číslo, ktoré určuje pozíciu nájdeného reťazca v položke. Ak sa reťazec nenájde, funkcia vráti nulu. Toto je skutočné jadierko procedúry. V prípade existencie reťazca nasledujú výpisy na obrazovku.

Iste ste zistili, že funkcia *pos* hľadá reťazec na ľubovoľnom mieste položky, nie len od začiatku. Teda ak máme priezviská Nový, Novák, Kovár, Šutek, Ovarská, Príbal, Šusteková a zadáme vyhľadávací reťazec „ov“, vypíšu sa priezviská N**ov**ý, N**ov**ák, K**ov**ár, Šustek**ov**á, ak sprísnime kritérium na "Ov", vypíše sa iba **Ov**arská.

Aby sme mohli použiť procedúru vyhľadávania, musíme patrične upraviť procedúru *Menu* a kontrolu stlačenia voľby položky menu v hlavnom tele programu. Pod bodom 4 je vyhľadávanie podľa priezviska, teda voláme procedúru *Vyhladaj('priezvisko')*, v bode 5 zase *Vyhladaj('cislo')*. Podrobnejší komentár sa nachádza priamo na listingu. Ospravedlňujem sa, že som v minulých častiach zvolil nevhodné meno pomocnej premennej *pos.* Aby nedochádzalo k zámene s funkciou *pos* v TP, premennú premenujeme na *pozit*.

Asi nikto nepochybuje o ďalšej veľmi dôležitej funkcii každého programu, a to aby svoje výsledky prezentoval nielen na obrazovke, ale aj na papieri. Preto ani nášmu programu nebude táto funkcia chýbať.

Výpisy na tlačiareň

Aby sme mohli pracovať s tlačiarňou, musíme používať knižnicu *printer*, ktorá sa podobne ako *crt* nachádza v súbore *TURBO.TPL*. Stačí ju dopísať do klauzuly *uses*, čím ju sprístupníme nášmu programu. Práca s tlačiarňou je veľmi zaujímavá, pretože dáva hmatateľné výsledky programu. Zatiaľ čo výsledky na obrazovke môžeme iba sledovať a maximálne zhodnocovať, výsledky vystupujúce na papier môžeme kedykoľvek dať k vyhodnoteniu alebo posúdeniu niekomu inému v ľubovoľný čas. Podmienkou je, že máme prístup k tlačiarni. (Ako tlačiareň funguje, podrobnejšie v seriáli „Slovenčina v počítači“, časť tretia, venovaná tlačiarni – pozri staršie čísla PC revue).

Najčastejšie používaným príkazom je *write* a *writeln*. Je veľmi podobný príkazu pre výpis na obrazovku, má však o jeden parameter viac. Tým parametrom je *lst*, takže zatiaľ čo výpis premennej *priatel.meno* na obrazovku vykoná príkaz *Write(priatel.meno)*, výpis na tlačiareň sa vykoná príkazom *Write(lst,priatel.meno)*. Jednoduché, však!

V závere deklaračnej časti programu sú popísané všetky procedúry spojené s tlačiarňou. Procedúry *Tlac_Hlavicky* a *Tlac_Hlavicky_Najdene* vypíšu hlavičku na tlačiarni. Procedúra *Tlac_Vypisu* zabezpečuje výstup nájdených viet, v ktorých sa nachádza hľadaný reťazec, na tlačiareň. Je jednoduchšia ako procedúra *Vypis.* Nemusíme nastavovať okná ako na obrazovke a nie je ani potrebné zabezpečovať úpravu pri vyššom počte riadkov, lebo zatiaľ čo obrazovka má max. 25 riadkov, papier je (relatívne) nekonečný. Takisto procedúra *Tlac_Vyhladaj* je podobná procedúre *Vyhladaj* a má obdobné znaky úpravy. (Pozri listing). V tlačových procedúrach však nefunguje príkaz *GotoXY*, pretože ho tlačiareň nepozná. Musíme ho nahradiť príkazom

write(lst,' ':17 – length(priatel.meno)));

Tento zabezpečí, že na tlačiarni sa vypíše taký počet medzier, koľko je 17 – dĺžka mena. Príklad: Ak je dané meno Jozef, toto sa vypíše na papier, za ním 17 – 5 medzier. Najbližší výpis začne na 5+17-5+1= 18. pozícii od okraja tlače. Takto dostaneme úhľadný výpis do stĺpcov na papier. Pre lepšie pochopenie môžeme zmeniť tieto hodnoty a odskúšať !

Upravíme menu, dopíšeme položky a nezabudneme na kontrolu stlačenej klávesy v hlavnom tele programu.

Teraz má náš program 4 nové funkcie, spolu deväť. Tu sú:

1*/ *Tvorba novej databáze
2*/ *Doplnenie databáze
3*/ *Výpis obsahu databáze
4*/ *Vyhľadávanie podľa priezviska
5*/ *Vyhľadávanie podľa čísla
6*/ *Tlač výpisu
7*/ *Tlač vyhľadaného priezviska
8*/ *Tlač vyhľadaného čísla
9*/ *Koniec

Prepíšte si program TELEFON4.PAS a preložte. Krokovaním F7 a F8 sledujte činnosť jednotlivých častí. Možete zmeniť niektoré hodnoty a opäť krokovať.

Precvičovacie otázky

1/ Čo *spôsobí ^g* v príkaze *Write(lst,^g)* ?
2/ Keby sme chceli pridať ďalšie dve položky do menu, aké poradové číslo by dostali?
3/ Upravte program pre vyhľadávanie bez rozdielu položky záznamu.

```
program Telefon4;    {program na tvorbu telefónneho zoznamu}
uses crt;                    {kvôli práci s obrazovkou}
{------------------ Deklaračná časť ----------------------}
type                         {deklarovanie typov premenných}
osoba = record               {údaje o osobe}
meno,
priezvisko : string[15];
cislo : string[10];          {tu je zmena na string}
end;
var                          {deklarovanie premenných}
priatel : osoba;             {údaje o priateľoch}
zoznam : file of osoba;      {súbor, kde sa budú ukladač údaje}
kolko : integer;     {pomocná premenná pre počet záznamov}
size : longint;              {prem. veľkosti súboru}
co : char;                   {pomoc. prem. pre načít. klávesnice}
hladane : string;            {hľadaný reťazec}

procedure Menu;              {voľby možností}
begin
clrscr;                      {vyçistenie obrazovky}
Writeln('1/ Tvorba novej databáze');
Writeln('2/ Doplnenie databáze'); {výpisy položiek menu}
Writeln('3/ Výpis obsahu databáze');
Writeln('4/ Vyhladávanie podla priezviska');
Writeln('5/ Vyhladávanie podla čísla');
Writeln('9/ Koniec');
end;
procedure Zadavanie;    {jednotná procedúra pre položky 1 a 2}
begin
Clrscr;                      {zmaže obrazovku}
repeat                       {opakuj...}
Clrscr;
Write('Zadaj meno  ');       {napíš 'Zadaj meno'}
Readln(priatel.meno);        {prečítaj zadané meno s klávesnice}
                             {a ulož do premennej priatel}
Write('Zadaj priezvisko  ');
Readln(priatel.priezvisko);
Write('Zadaj cislo  ');
Readln(priatel.cislo);
if priatel.cislo<>'koniec' then
write(zoznam,priatel);       {uloženie údajov z premennej priatel}
                             {do súboru zoznam}
until priatel.cislo = 'koniec';    {...pokým nie je zadané "koniec"}
end;
```

```
function Nie_je_subor : boolean;    {kontrola existencie súboru}
begin
Nie_je_subor:=false;          {nastavenie na "nepravda"}
{$I-}
Reset(zoznam);                {otvorenie už existujúceho súboru}
{$I+}
if IOResult <> 0 then         {kontrola vstupno výstupnej operácie}
begin                         {ak je chyba – súbor neexistuje}
Gotoxy(10,10);                {tak hlásenie}
Writeln('Datový súbor neexistuje ! Založte nový !',^g);
repeat until keypressed;      {čaká na stlačenie ľub. klávesy}
Nie_je_subor:=true;           {nastavenie, že súbor nie je}
end
else
Nie_je_subor:=false;          {inak súbor je}
end;
procedure Novy;               {vytvorí nový súbor}
begin
Rewrite(zoznam);              {vytvorenie a otvorenie súboru}
ZADAVANIE;                    {zapisuj}
Close(zoznam);                {uzavri súbor na disku}
end;
procedure Pridaj;             {otvorí a pridá položky}
begin                         {k už existujúcemu súboru}
if Nie_je_subor = true then Exit;  {ak súbor nie je, tak koniec
bloku}
size:=FileSize(zoznam);       {zistenie veľkosti súboru}
seek(zoznam,size);            {choď na koniec súboru}
ZADAVANIE;                    {a dopisuj}
Close(zoznam);                {zavri súbor}
end;
procedure Hlavicka;           {vytvorí hlavičku výpisu}
begin
Window(1,1,80,25);
Writeln('*************************************************');
Writeln;
Writeln('Meno       Priezvisko      Telefón  ');
Writeln('------------------------------------------------------------');
end;
procedure Hlavicka_Najdene;             {vytvorí hlavičku výpisu}
begin
Window(1,1,80,25);
Writeln('              Výpis nájdených záznamov ');
Writeln('*************************************************');
Writeln;
Writeln('Meno       Priezvisko       Telefón  ');
Writeln('------------------------------------------------------------');
end;
procedure Vypis;              {prezeranie zápisov}
var pozit : integer;          {premenná, určujúca pozíciu na obr.}
ch : char;                    {pom.prem.pre načítavanie klávesnice}
begin
if Nie_je_subor = true then Exit;  {kontrola existencie súboru}
Clrscr;
Hlavicka;                     {volanie procedúry Hlavicka}
kolko:=0;
pozit:=0;                     {nastavenie premenných}
Window(1,6,80,25);            {nastavenie okna obrazovky}
repeat
Read(zoznam,priatel);         {čítaj údaje zo súboru do premennej}
pozit:=pozit+1;               {zväč počítadlo o jeden}
kolko:=kolko+1;
GotoXY(1,pozit);              {nastav kurzor na stanovenú pozíciu}
Write(priatel.meno);          {a vypíž údaj o mene z premennej}
GotoXY(18,pozit);
Write(priatel.priezvisko);
GotoXY(35,pozit);
Writeln(priatel.cislo);
if kolko mod 19 = 0 then      {ak počet záznamov je násobok 19, tak}
begin
pozit:=0;                     {vynuluj pozíciu na obr.}
write('Pokračovať? Stlač "a" alebo "A" pre pokračovanie,inak
koniec výpisu',^g);
                              {výpis na obrazovku}
ch :=readkey;                 {načítaj klávesnicu}
if ((ord(ch) = 97) or (ord(ch) = 65)) then
                              {ak bolo stlačené "a" a "A",}
clrscr                        {tak zmaž obr.}
else begin                    {inak}
clrscr;                       {zmaž obr.}
Window(1,1,80,25);{nastavenie nového okna}
exit;                         {koniec bloku}
end;
end;
until Eof(zoznam);            {...dokiaľ nie je koniec súboru}
Writeln;                      {prázdny riadok}
Writeln('Počet záznamov v súbore :',kolko); {hlážka o počte
záznamov}
Readln;                       {čakám na stlačenie Enteru}
Close(zoznam);                {zavri súbor}
Window(1,1,80,25);            {nastavenie nového okna}
end;
procedure Vyhladaj(podla_coho : string); {vyhľadávanie zada-
ného reťazca}
var pozit : integer;          {premenná, určujúca pozíciu na obr.}
ch : char;                {pom. prem. pre načítavanie klávesnice}
kde : string;
begin
if Nie_je_subor = true then Exit; {kontrola existencie súboru}
GotoXY(10,10);                {presun kurzoru}
if podla_coho = 'priezvisko' then Write('Zadaj priezvisko : ');
if podla_coho = 'cislo' then Write('Zadaj číslo : ');
Readln(hladane);              {načítanie hľadaného reťazca}
Clrscr;
Hlavicka_Najdene;             {volanie procedúry
Hlavicka_Najdene}
kolko:=0;
pozit:=0;                     {nastavenie premenných}
Window(1,6,80,25);            {nastavenie okna obrazovky}
repeat                        {opakuj...}
Read(zoznam,priatel);         {čítaj údaje zo súboru do premennej}
if podla_coho = 'priezvisko' then kde := priatel.priezvisko;
if podla_coho = 'cislo' then kde := priatel.cislo;
if (pos(hladane,kde) >0 ) then
begin                         {tak}
pozit:=pozit+1;               {zväč počítadlo o jeden}
kolko:=kolko+1;
GotoXY(1,pozit);              {nastav kurzor na stanovenú pozíciu}
Write(priatel.meno);          {a vypíž údaj o mene z premennej}
GotoXY(18,pozit);
Write(priatel.priezvisko);    {o priezvisku}
GotoXY(35,pozit);
Writeln(priatel.cislo);       {o čísle}
if kolko mod 19 = 0 then      {ak počet záznamov je násobok 19, tak}
begin
pozit:=0;                     {vynuluj pozíciu na obr.}
write('Pokračovať ? Stlač "a" alebo "A" pre pokračovanie,inak
koniec výpisu',^g);
                              {výpis na obrazovku}
ch :=readkey;                 {načítaj klávesnicu}
if ((ord(ch) = 97) or (ord(ch) = 65)) then
                              {ak bolo stlačené "a" a "A",}
clrscr                        {tak zmaž obr.}
else begin                    {inak}
clrscr;                       {zmaž obr.}
Window(1,1,80,25);            {nastavenie nového okna}
exit;                         {koniec bloku}
end;
end;
end;
until Eof(zoznam);            {...dokiaľ nie je koniec súboru}
Writeln;                      {prázdny riadok}
Writeln('Počet nájdených záznamov v súbore :',kolko); {hlážka
o počte záznamov}
Readln;                       {čakám na stlačenie Enteru}
Close(zoznam);                {zavri súbor}
Window(1,1,80,25);            {nastavenie nového okna}
end;
{-------------------- Príkazová časť ---------------------}

BEGIN
Assign(zoznam,'telefon.zoz');      {priradenie mena súboru}
repeat                        {opakuj...}
MENU;                         {vypíž menu}
repeat                        {opakuj}
co:=readkey                   {čítaj klávesnicu}
```

```
until co in ['1','2','3','4','5','9'];
                              {pokiaľ nebolo stlačené "1" alebo "2" }
                              {alebo "3" alebo "4"....}
if co = '1' then Novy;        {ak je to 1.,tak zadávanie údajov}
if co = '2' then Pridaj;      {ak je to 2.,tak pridávanie údajov}
if co = '3' then Vypis;       {ak je to 3.,tak výpis súboru}
if co = '4' then Vyhladaj('priezvisko'); {ak je to 4.,tak vyhľadá-
vanie}
                              {podľa priezviska}
if co = '5' then Vyhladaj('cislo');  {ak je to 5.,tak vyhľadávanie}
                              {podľa čísla}
until co = '9';               {pokial si nezadal 9.}
END.                          {čo je koniec celého programu}
```

Program je už pomerne zložitý. Krokovanie F7 a F8 podáva iba najzákladnejšie informácie. My by sme chceli vedieť, čo sa deje s ľubovoľnou premennou, a tiež by sme chceli, aby bol trošku farebnejší. Veď mnohí máme farebné monitory. O tom nabudúce.

STRETNUTIE S PASCALOM

6. časť

Opakovanie

Náš program sa pekne rozrastá. Pridali sme do neho dve veľmi dôležité funkcie každej databanky: vyhľadávanie a výpis na tlačiareň. A to ho ešte chceme „ofarbiť“. Ale najprv si skontrolujeme

Domáce úlohy

1/ Samozrejme, že ste na to prišli ihneď. Ak Write(^g) „zapípa“ na počítači, tak Write(lst,^g) rozozvučí bzučák tlačiarne.
2/ V položkovaní hlavného menu sme už použili všetky čísla okrem nuly. Viacmiestne čísla napr. 11,12 a iné to byť nemôžu, pretože program reaguje iba na jedno stlačenie klávesu. Mohli by sme síce ďalšej položke priradiť nulu akoby desiatku, avšak výhodnejšie by bolo použiť písmená abecedy. Potom musíme upraviť kontrolu klávesov, ale čo je najdôležitejšie, ošetriť, aby program reagoval na stlačené malé „a“ aj na veľké „A“. Vyriešenie je v procedúre Vypis, ale existujú aj iné spôsoby.
3/ V tomto prípade musíme zlúčiť oblasti vyhľadávania. Spôsobov je viac, vyskúšajte logický súčet OR.

Farby v Turbo Pascale

Ak chceme v programe riešiť farby, musíme si povedať niečo o „farebnosti“ Turbo Pascalu a počítača vôbec.

Farebnosť je ovplyvnená mnohými činiteľmi a preto len stručne:

Farby môžu zobrazovať iba grafické karty typu CGA, EGA a VGA. Samozrejmou podmienkou je farebný monitor príslušný k tej-ktorej grafickej karte. V textovom režime je možné využiť 16 základných farieb. (Znovu upozorňujem! Nemýliť s farbami vo Windows a pod.). Pozrime sa na to z pohľadu Turbo Pascalu:

Celá farebná škála je určená štandartnou premennou TextAttr, ktorá je typu byte. Bity 0 až 3 určujú farbu znaku, bity 4 až 6 jeho pozadie. Siedmy bit určuje blikanie (blink). Z tohoto vidíme, že znak môže mať všetkých šestnásť farieb, zatiaľ čo jeho pozadie iba prvých osem (0-7). Ale ktoré farby? Tu sú, vrátane anglických názvov, pretože tie sú zároveň preddefinovanými konštantami v TP:

0 = black	– čierna
1 = blue	– modrá
2 = green	– zelená
3 = cyan	– tyrkysová
4 = red	– červená
5 = magenta	– fialová
6 = brown	– hnedá
7 = lightgray	– svetlošedá
8 = darkgray	– tmavošedá
9 = lightblue	– svetlomodrá
10 = lightgreen	– svetlozelená
11 = lightcyan	– svetlotyrkysová
12 = lightred	– svetločervená
13 = lightmagenta	– svetlofialová
14 = yellow	– žltá
15 = white	– biela

Na prácu s farbami slúžia procedúry **TextBackground, TextColor, NormVideo, HighVideo** a **LowVideo**. Tieto však nemusia bežnému programátorovi stačiť a tak si niektoré upraví. Upozorňujem, že s farbami sa dajú robiť rôzne „psie kusy“, ale na to isto prídete sami. Jednu úpravu najdete aj na listingu č.4. Veď prečo mať len osem farieb pre pozadie, keď ich môžeme mať 16 (na úkor blinku).

Teraz už môžeme náš program ofarbiť. A taktiež by nemuselo byť vidieť blikajúci kurzor, keď ho zrovna nepotrebujeme. Hm, ak však pridáme do zdrojového textu tieto potrebné úpravy, začne byť zdrojový text neprehľadný. Najlepšie by bolo, keby definície procedúr týchto úprav boli uložené niekde inde a my ich len v našom programe pekne volali, tak ako je to napr. s procedúrou ClrScr z jednotky **CRT**. Ale ako na to? No predsa cez

Unity

Od verzie 5 má programátor v TP možnosť vytvárať vlastné skupiny podporných procedúr a funkcií, ktoré môže kedykoľvek využívať tak, že už len zavolá meno funkcie či procedúry z danej skupiny.

Táto skupina sa nazýva **unit**. V angličtine síce znamená „jednotka“, ale vystižnejší názov u našich programátorov je „knižnica“. (Veď oni sa u Borlandu neskôr spamätali a potom zavedli aj knižnice = library, práve kvôli Windows). Preto aj my budeme unitu hovoriť knižnica. (Puntičkári prepáčia!).

Unity sú veľmi dôležitá súčasť programovania v TP. Bez nich napísať slušnú aplikáciu snáď ani nejde. V objektovo orientovaných programoch sú unity základom, lebo hlavná časť zdrojového textu máva spravidla iba tri (slovom tri) riadky, zvyšok sa odohráva práve v unitoch. Preto teraz odbočíme a povieme si, čo to tie unity vlastne sú. Predstavme si takúto situáciu: Určité procedúrky používame vo svojich programoch veľmi často, napríklad pípnutie počítača a tlačiarne. Doteraz sme museli procedúry definovať v každom programe. Aj keď už boli raz vymyslené, musíme ich zakaždým pracne opisovať. Čo tak ich vložiť do nejakej knižnice, kde budú pekne raz a navždy nadefinované, no a my ich budeme už len používať. Dokonca môžeme časom zabudnúť, ako sme ich vytvorili (veľmi často, dámy a páni!), stačí, ak niekde budeme mať odložený ich názov a popis parametrov. Štruktúra unitu je na listingu č.1. Vidíme, že sa skladá z troch častí. V časti **INTERFACE** sa deklarujú konštanty, typy, premenné, funkcie a procedúry, ktoré sú verejné (public), t.j. sú dostupné každému užívateľovi tejto knižnice. Užívateľ má k týmto entitám prístup, akoby boli deklarované v jeho samotnom programe. Stačí, aby vedel iba ich názov, na čo slúžia a poprípade význam ich parametrov. Ako je však tá-ktorá procedúra urobená, je nadefinované v časti **IMPLEMENTATION**. Táto časť je neverejná (private), teda nemá byť prístupná pre iného užívateľa okrem autora. Toto je veľmi dôležité! Programátor vám môže urobiť unit, ale ak nechce (z rôznych a celkom pochopiteľných dôvodov, trh je trh), aby ste vedeli, ako tie procedury napísal, dodá vám unit v preloženom tvare a iba opis interfejsovej časti, aby ste vedeli, čo unit dokáže. V implementačnej časti môžu byť nadefinované aj iné, interné konštanty, typy, premenné, funkcie a procedúry, ktoré sú potrebné pre činnosť unitu a nie sú popísané v časti interface. Tretia – **inicializačná časť** – zinicializuje niektoré premenné a procedúry, nemusí to byť vždy nutnosťou, potom je táto časť prázdna. Náš konkrétny unit si teraz vytvoríme. Jeho popis je na listingu č. 2. Unit sa píše po-

dobne ako samotný program. V hlavičke unitu je kľúčové slovo **unit** a za ním meno unitu (Zvonky). Na rozdiel od programu je slovo unit a meno povinné. **Meno musí byť jedinečné a musí sa zhodovať s menom** súboru (ZVONKY.PAS), inak prekladač vyhlási chybu! Následuje kľúčové slovo **INTERFACE**, ktorým začína verejná časť. Použijeme jednotku Printer (z TURBO.TPL) definíciou *uses Printer* a zadeklarujeme konštantu *Bell* a hlavičky procedúr *procedure Beep(kolko:byte);* s parametrom kolko typu byte a *procedure PrnBeep;*. Samotné konštrukcie týchto procedúr sa nachádzajú až v privátnej časti za kľúčovým slovom **IMPLEMENTATION**. Vydefinujeme procedúry, prvá je tzv. parametrická, kde parameter kolko určuje počet pípnutí. Slovami **BEGIN** a **END**. je ohraničená **inicializačná časť**, ktorá je v našom prípade prázdna, lebo je unit veľmi jednoduchý. Takto napísaný unit (samozrejme v prostredí TP) uložíme na disk pod menom **ZVONKY.PAS**. Voľbou *Compile – Compile* s nastavením *Destination – Disk* našu prvú knižnicu zkompilujeme. Preložením dostaneme tvar **ZVONKY.TPU** (**T**urbo **P**ascal **U**nit). Z tohoto tvaru už nikto nevyčíta, ako sú jednotlivé procedúry vytvorené. Ak chcete túto knižnicu dopriať priateľom, ale nechcete, aby vedeli, ako je napísaná, dajte im preložený tvar a opis interfejsovejčasti po kľúčové slovo IMPLEMENTATION. Býva dobrým zvykom túto časť uložiť ako súbor s príponou .INT, aby užívateľ vedel, aké procedúry a ich parametre sú obsiahnuté v unite. (Len pre zaujímavosť: unit v tvare .TPU a súbor .INT sa môže voľne šíriť a používať, ak to ovšem autor dovolí. Ale ak chcete vedieť, ako autor jednotlivé funkcie a procedúry unitu vytvoril, musíte si od neho zakúpiť zdrojové texty, no a tie už niečo stoja). Našu novú knižnicu teraz vyskúšame. Na listingu č.3 je program SKUSKA. Všimnime si veľmi dôležitú vec! V klauzulke uses Zvonky sme zavolali náš unit a to je všetko! V hlavnom tele programu ZVONKY stačí už len volať mená procedúr Beep a PrnBeep. Samozrejme, že môžeme našu knižnicu rozširovať alebo si napíšeme inú. Unity sa môžu vnorovať, teda unit môže používať iné unity. Skúste si sami napísať svoj unit a pohrajte sa s ním.

Cítil som potrebu venovať sa unitom, keďže sú jedným z najdôležitejších modulov programovania v TP. Teraz tieto vedomosti a praktické skúsenosti využijeme pri našej telefónnej databanke. Na listingu č.4 je výpis unitu MTOOLS (m-nástroje). Je to už skutočná knižnica, ktorá nášmu hlavnému programu TELEFON5 (listing č. 5) poskytuje celkom slušné možnosti. Pozrime sa podrobnejšie na verejnú časť (INTERFACE) unitu MTools. Spoznávame konštantu Bell a procedúry Beep a PrnBeep. Pribudli deklarácie typov Colors (farby) a BorderType (typ rámčeku). V konštantách sú jednotlivé vzory rámčekov definované postupnosťou: ľavý horný roh (TL), pravý horný roh (TR), ľavý dolný roh (BL), pravý dolný roh (BR), vodorovná čiara (FH) a zvislá čiara (FV). Takýchto vzorov je šesť. V časti var sú zadefinované premenné tejto knižnice. Podstatné sú pre nás deklarácie jednotlivých funkcií a procedúr. Ich význam je podrobne popísaný v komentároch zdrojového textu. V sekcii IMPLEMENTATION sú tieto funkcie a procedúry nadefinované. Ako sme si povedali, táto časť je privátna, a je dôležitá iba pre autora. Aby ste mohli túto knižnicu použiť v našom seriáli, je uvedený kompletný výpis zdrojového textu. Z určitých dôvodov nebudeme túto časť teraz vysvetľovať podrobnejšie, stačia priložené komentáre. Prepíšte si ho, uložte ako MTOOLS. PAS a skompilujte do tvaru MTOOLS.TPU. Praktické použitie knižnice je v programe TELEFON5.PAS na listingu č. 5. Pozorne upravte zdrojový text z predchádzajúcej časti seriálu. Postupným ladením a krokovaním preverte činnosť jednotlivých nových procedúr. Keď v zdrojovom texte krokovací pás príde na novú procedúru a stlačíme F7, na obrazovke sa automaticky objaví zdrojový text unitu MTOOLS.PAS a krokovanie bude prebiehať v ňom. Po ukončení tela procedúry sa krokovací pás vráti do hlavného zdrojového textu. Zložité? No, odskúšajte, isto sa to objasní. Vzhľadom na už určité získané skúsenosti nebudeme preberať TELEFON5.PAS podrobne.

Domáca úloha

bude iba jedna. Preštudujte poriadne implementačnú časť MTOOLS. PAS. Nebojte sa experimentovať! Predtým ako „siahnete“ do unitu, nezabudnite si urobiť zálohy! Využijte Help priamo v TP, alebo sa pozrite do dostupnej literatúry. Len takto pochopíte presne jeho činnosť a bude vás inšpirovať k novým úpravám.

Tak, náš program funguje, vypisuje, tlačí, pípa, vyhľadáva a je aj celkom farebný. Ale stále to ešte nie je ono, a preto ho budeme naďalej vylepšovať. Ale to až nabudúce.

Listing č. 1

```
{direktívy prekladača}
unit meno_unitu

INTERFACE

uses jednotky

***************** deklaračná časť *****************
TYPE

CONST

VAR

FUNCTION                    {len hlavička s parametrami}

PROCEDURE                   {len hlavička s parametrami}

***************** definičná časť ********************
IMPLEMENTATION
FUNCTION meno_funkcie;
begin
  .....
end;
PROCEDURE meno_procedury(parametre);
begin
  .....
end;

***************** inicializačná časť ******************
BEGIN
  .....
END.
```

Listing č. 2

```
unit Zvonky;        {zabezpečuje zvukovú signalizáciu v PC alebo
tlačiarni}

INTERFACE           {interfejsová  - verejná - public časť}

uses printer;       {kvôli výstupu na tlačiareň}

const             {deklarácie - konštanty : }
   Bell = #$7;              {            - znak z ASCII tabuľky}

procedure Beep(kolko:byte);    { - a procedúr: - zapípa v počítači,
parameter}
                        {              kolko určuje počet pípnutí}
procedure PrnBeep;            {            - zapípa na tlačiarni}

(*************)

IMPLEMENTATION        {implementa‡n  - súkromná - private
časť}

procedure Beep(kolko:byte);       {definícia procedúry Beep}
var i : byte;                   {pomocná  premenná  pre cyklus for -
to}
begin
  for i:=1 to kolko do write(Bell);{pípne "kolko"-krát}
end;

procedure PrnBeep;    {definícia procedúry PrnBeep}
begin
  write(lst,Bell);
end;
```

```
(***************)
BEGIN                {inicializačná  časť – v tomto prípade prázdna}
END.
```

Listing č. 3

```
program Skuska;          {overenie činnosti unitu Zvonky}
uses Zvonky;             {zavolanie našej knižnice}

BEGIN
  Beep(3);               {procedura Beep z knižnice Zvonky s
parametrom 3, teda pípne tri-krát}
  PrnBeep;               {pípne na tlačiarni}
END.
```

STRETNUTIE S PASCALOM

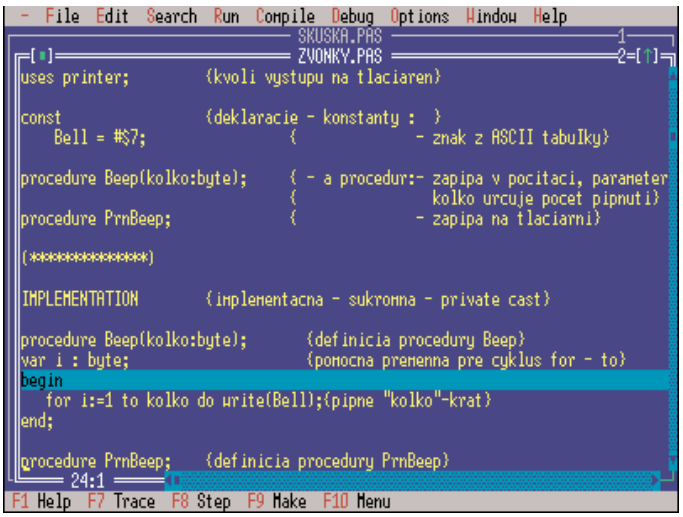
7. časť

Opakovanie

Už vieme, čo sú to unity, na čo slúžia a ako sa tvoria. Najdôležitejšie je vedieť, ako použiť unity od iných autorov.

A že sa dá v Turbo Pascale napísať všetko (teda skoro všetko), o tom vás už hádam ani presvedčovať nemusím.

Náš program prešiel mnohými zmenami. Ešte stále je možné ho vylepšovať, ale na to treba určité skúsenosti a nápady. O tom, že nápady máte, a dobré, svedčia vaše listy na moju adresu. A skúsenosti sa dostavia praxou. Sú však úplne začínajúci programátori, ktorí by potrebovali čo-to objasniť z techniky odladovania programov. A preto dnes nebudeme programovať v pravom slova zmysle, ale povieme si niečo o tej náročnejšej stránke tvorby produktov. A k tomu nám poslúži toto



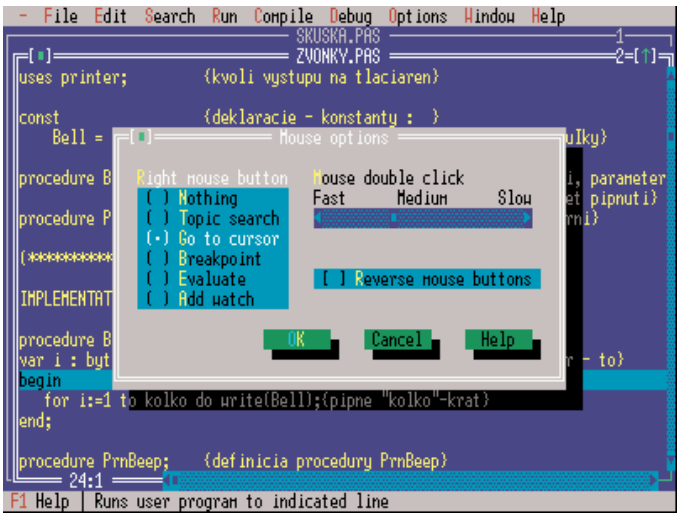
Obr. 1

INTERMEZZO

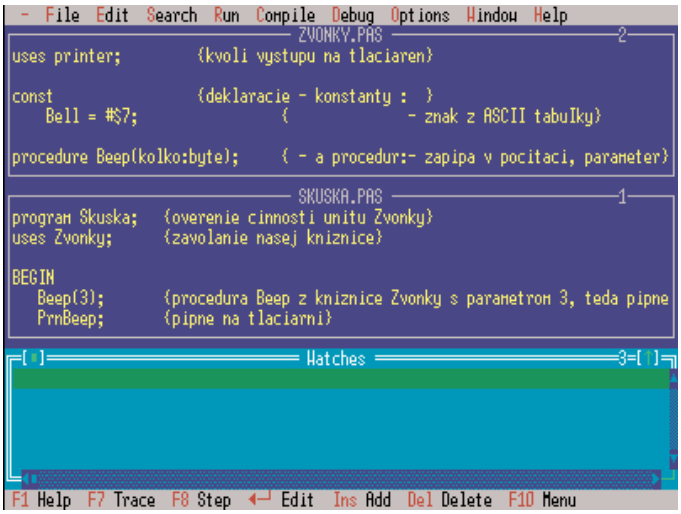
kde si povieme o ďalších, výkonejších metódach odladovania programov v prostredí IDE. Turbo Pascal má vstavaný tzv. debugger (angl. „odvšivovač“), ktorý práve napomáha odstraňovať závady a chyby v programe. Je to veľmi mocný ladiaci prostriedok, preto si najdôležitejšie funkcie vysvetlíme.

Krokovanie a trasovanie – Step over a Trace into

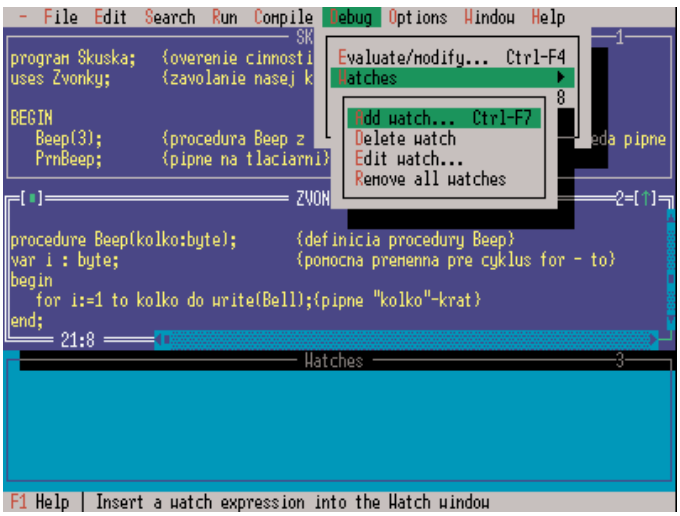
Krokovanie programu voľbou **Step over** (F8) a trasovanie voľbou **Trace into** (F7)už poznáme. Len pre zopakovanie: Step over vykonáva iba riadky hlavného tela programu. Aj procedúru identifikuje ako jeden príkaz, a preto ju vykoná ako celok. Ak chceme krokovať vo vnútri procedúry, použijeme Trace Into (F7), čím sa dostaneme do tela procedúry. Podmienkou je, že na disku sa nachádza súbor so zdrojovým kódom danej procedúry. Ak je procedúra volaná z unitu, interný debugger zavolá zdrojový text unitu (ak existuje) a nastaví krokovanie v tele unitu (obr. 1).



Obr. 2



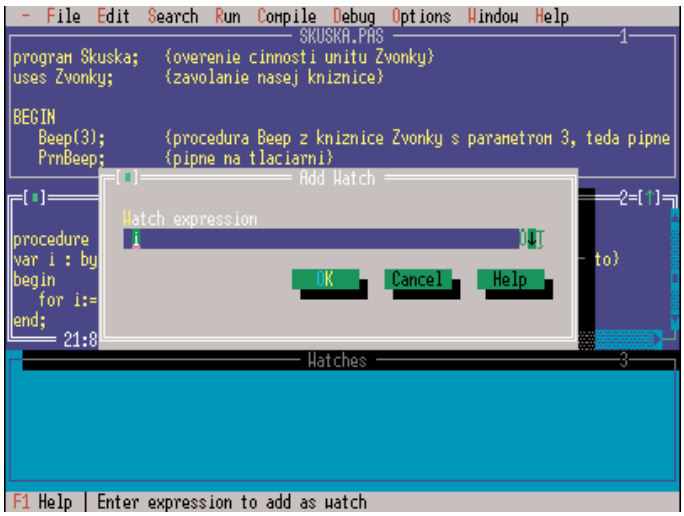
Obr. 3



Obr. 4

Choď na kurzor – Go to cursor

Ďalšou veľmi dobrou vlastnosťou debuggeru je možnosť preskočiť tú časť programu, ktorá je odladená a pokračovať v kro-

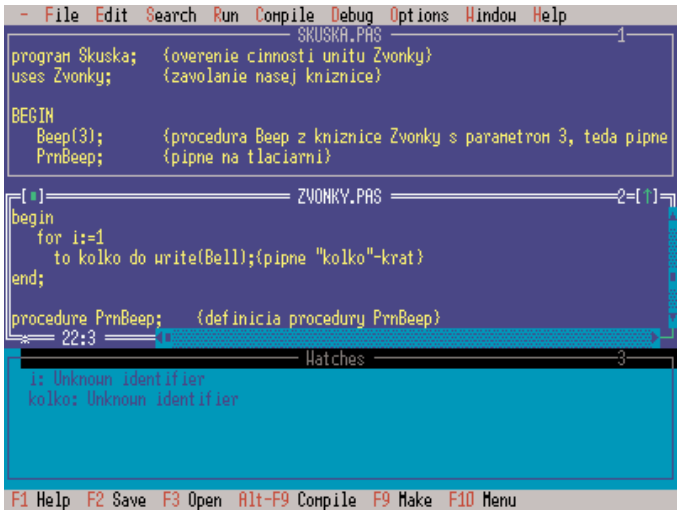


Obr. 5

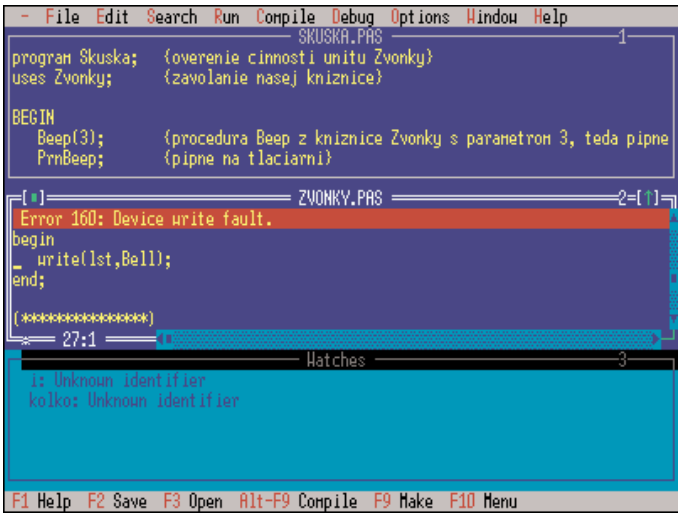
ich postupným krokovaním. TP umožňuje nastaviť túto možnosť voľbou **Option – Environment – Mouse – Go to cursor** (obr. 2) tak, že stačí v texte kliknutím ľavého tlačítka myši umiestniť kurzor a kliknutím pravého tlačítka nahradiť stlačenie F4. Nastavte a vyskúšajte.

Sledovanie – Watch

Asi najefektívnejším spôsobom odladovania programu je sledovanie hodnôt premenných, funkcií a výrazov v okne **Watches**. Okno otvoríme takto: Voľbou **Window – Watch** otvoríme okno Watches (Sledovania). Voľbou **Window – Tile** usporiadame okná na obrazovke tak, aby bolo vidieť okná vedľa seba (obr. 3). Kurzor postavíme na premennú, ktorú chceme sledovať v okne Watches. Voľbou **Debug – Watches – Add watch** (obr. 4) alebo jednoduchšie stlačením „horúcich klávesov“ Ctrl-F7 sa na obrazovke sa zobrazí okno Watch expression (obr.5) a potvrdením OK sa vloží sledovaná premenná do okna Watches. Takto si zobrazíme aj ostatné premenné, ktoré chceme sledovať (obr. 6). Medzi jednotlivými oknami na obrazovke môžeme prechádzať

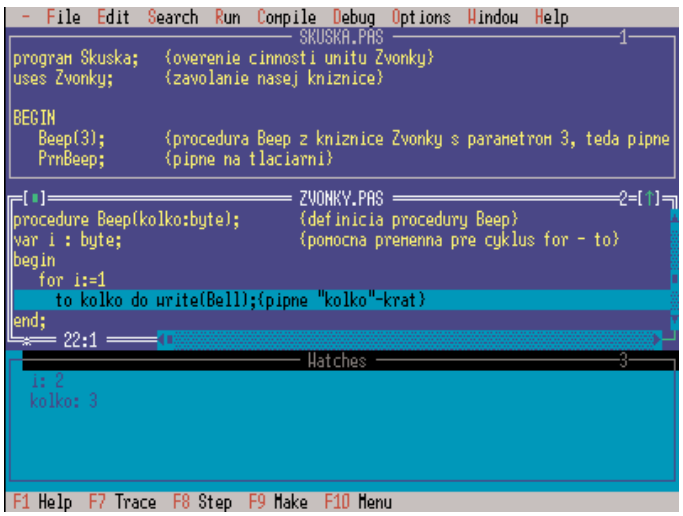


Obr. 6

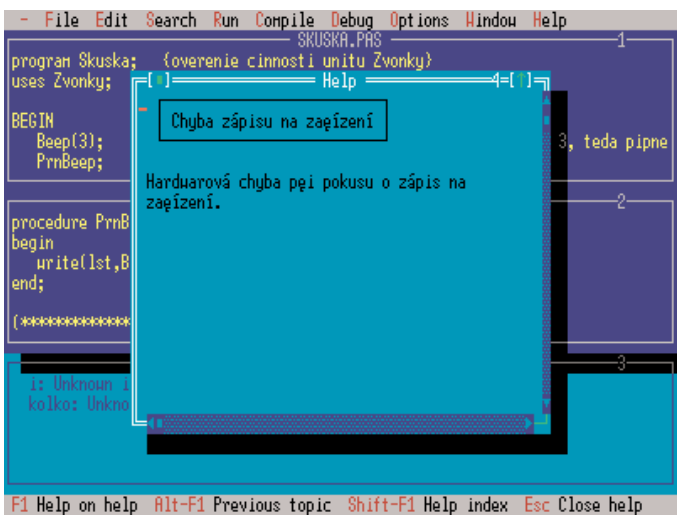


Obr. 8

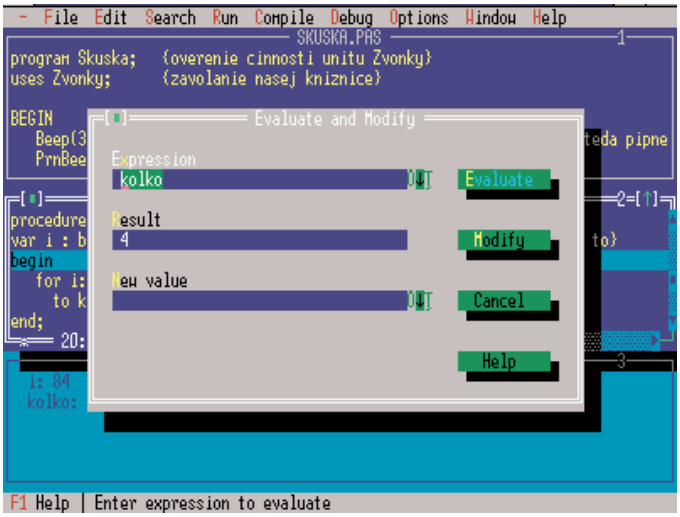
kovaní od určitého riadku. Toto umožňuje funkcia **Go to cursor**. V zdrojovom texte programu umiestnime kurzor na riadok, od ktorého chceme začať krokovať. Voľbou **RUN – Go to cursor** alebo stlačením F4 sa spustí kompilátor, program sa vykoná až po riadok, na ktorom kurzor stojí. Túto možnosť využívame vtedy, keď predchádzajúce riadky zdrojového textu sú už odladené alebo absolútne bezchybné a preto sa nemusíme zdržiavať voľbou **Window – Next** alebo stlačením F6. A teraz môžeme spustiť samotné krokovanie. Všimnime si, že na začiatku je v okne Watches poznámka „Unknown identifier“ (neznámy identifikátor). To preto, že debugger ešte nepozná tieto premenné. Až po spustení programu, či už priamo voľbou RUN alebo krokovaním, kompilátor preloží zdrojový text a uloží premenné do pamäti počítača. Samozrejme, že ak krokujeme, debugger už pozná pre-



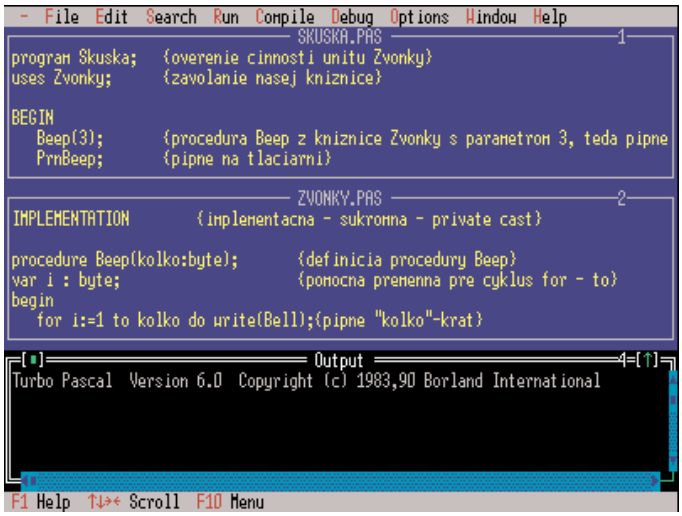
Obr. 7



Obr. 9



Obr. 10



Obr. 11

menné, ale nepozná ich hodnoty. Preto sa stáva, že v okne Watches sú premenným priradené náhodné hodnoty. Neľakajte sa, je to normálne! Akonáhle krokovanie pokročí na príkaz, kde je premennej priradená konkrétna hodnota, táto sa objaví v okne Watches (obr. 7). A práve od tohoto okamžiku je nutné pozorne sledovať stav premenných alebo výrazov v závislosti na zdrojovom kóde. Ak sa hodnoty premenných nemenia tak, ako je žiadúce, musíme vysledovať príčinu. Sledované hodnoty môžeme do okna Watches doplňať aj počas krokovania. Výhodné je nastavenie myši (obr. 2) tak, že stlačenie pravého tlačítka myši spôsobí funkciu Add watch (keď som už pri tej myši, tak ľaváci si zaškrtnutím voľby Reverse....môžu zmeniť zmysel tlačítok podľa seba.). V prípade, že v ladení programu nechceme pokračovať, ale krokovací pás svieti a nedošli sme až na koniec programu, ukončíme tieto činnosti voľbou **Run – Program reset – zastavenie a znovunastavenie programu** alebo stlačením Ctrl-F2. Táto skutočnosť **korektne** ukončí beh programu alebo ladenia, uzavrie všetky otvorené súbory, vyprázdni buffery a vyčistí zásobník. V opačnom prípade sa môže stať, že program zhavaruje a dôjde k nenapravitelným škodám. Hlavne sa to týka takých projektov, ktoré pracujú s databázovými súbormi. **Nekorektným ukončením behu programu alebo ladenia môže dôjsť ku strate dát alebo poškodeniu súborov!. Nikdy neukončujte ladenie inak, ako prekrokovaním celého programu alebo stlačením Ctrl-F2!**

Prerušenie programu – break – pri ladení programu (samozrejme v prostredí IDE) môže dôjsť ešte k jednej nepríjemnej veci, a to k **zacykleniu** programu. Vtedy nemožno prerušiť beh programu (procedúry, príkazu) inak, ako súčasným stlačením klávesov **Ctrl-Break**. Tým zabránime havárii programu a debugger nás po stlačení klávesu Escape vráti do editoru.

Chyby pri behu programu – Error

Ak počas behu alebo ladenia vznikne chyba v programe, vysvieti sa na vrchole okna červený pás s číslom a popisom chyby. Kurzor ukazuje na miesto výskytu chyby (obr. 8). Ak teraz stlačíme kláves F1, zobrazí sa help, popisújúci chybu a niektorých prípadoch napovie, ako ju odstrániť (obr. 9).

Vyhodnotenie a zmena – evaluate and modify

Ak chceme zistiť, ako bude program reagovať na inú hodnotu premennej, ako je nastavená programom, môžeme túto zmeniť. Voľbou Debug-Evaluate/modify alebo Ctrl-F4 sa zobrazí okno **Evaluate and Modify** (obr. 10). V kolónke Expression zadáme premennú alebo výraz, ktorého hodnotu chceme zistiť a zmeniť. Zadaním povelu Evaluate sa v kolónke Result objaví skutočná hodnota výrazu. Ak v kolónke New value napíšeme novú hodnotu, zadaním povelu Modify ju program akceptuje. takto môžeme zistiť, ako sa správa program pri hodnotách ktoré sú málo pravdepodobné, ale z konštrukcie algoritmu môžu nastať. Toto je veľmi dôležité hlavne pri takých programoch, ktoré očakávajú zadávanie dát užívateľom a následným výpočtom hodnoty premennej.

Užívateľská obrazovka – user screen

slúži na zobrazovanie výstupnej obrazovky programu. Činnosť na tejto obrazovke môžeme počas krokovania sledovať prepínaním klávesami Alt-F5. Ak chceme sledovať činnosť na obrazovke priamo z prostredia editoru, použijeme voľbu **Window – Output**. V dolnej časti editoru sa objaví okno Output (obr. 11). **Upozornenie! User screen zobrazuje ako textový, tak grafický výstup. Output zobrazuje iba textový výstup!**

Domáca úloha

Vyskúšajte si všetky tieto funkcie a prístupy. Bude dobré, ak si ich uvedomíte a dostanete ich do „krvi“. Neskôr, pri tvorbe veľkých projektov sa bez nich nezaobídete!

Zostala nám najdôležitejšia funkcia debuggeru – prerušenia – **breakpoints**. Tie si vysvetlíme inokedy.

STRETNUTIE S PASCALOM

8. časť

Opakovanie

V prvej časti intermezza sme si vysvetlili veľmi efektívne metódy odladovania programov v prostredí IDE. Len pre poriadok si zopakujme ich názvy a skrátené klávesy:

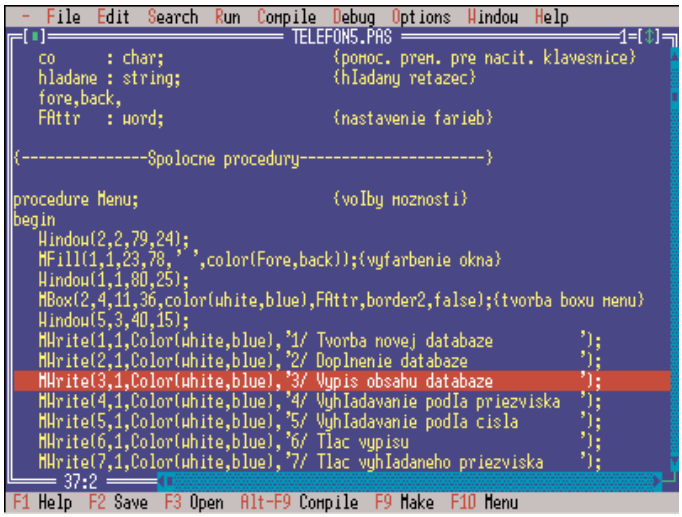
– Step over	F8
– Trace into	F7
– Go to cursor	F4
– Evalue/Modify	Ctrl-F4
– Watching	Ctrl-F7
– Run	Ctrl-F9
– Program reset	Ctrl-F2

Týmto skráteným klávesom hovoríme aj horúce, od anglického slova *hotkeys*. Naučte sa používať horúce klávesy, uvidíte, ako sa vaša práca zrýchli.

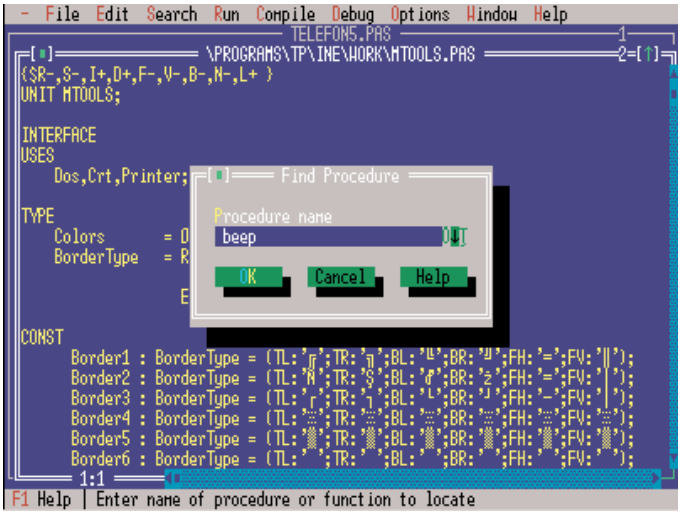
Najefektívnejším spôsobom odladovania programu, ktorý využíva všetky doteraz spomenuté prvky ladenia je

Používanie bodov prerušenia

alebo jednoducho *breakpoints*. Súčasne môžeme mať aktívnych až 16 bodov prerušenia. Slúžia k pozastaveniu programu a potom čaká na ďalšie ladiace inštrukcie.

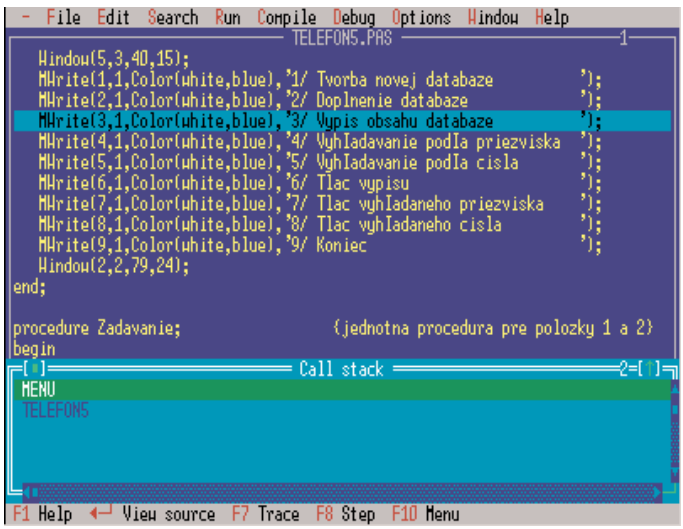


Obr. 1



Obr. 3

Body prerušenia sa v .EXE súbore po kompilácii už nevyskytujú, ale existujú iba počas ladenia. Nastavíme ich tak, že kurzor presunieme na riadok, kde chceme program pozastaviť, pričom tieto riadky by mali obsahovať aspoň jeden vykonateľný príkaz (nie prázdne riadky, komentáre, direktívy kompilátoru, konštanty, typy, návestia, deklarácie premenných alebo hlavičky program, unit, procedure, function) a zvolíme **Debug-Toggle Breakpoint** alebo Ctrl-F8. Riadok s takto nastaveným bodom prerušenia je zvýraznený čeveným pásom (obr. 1). Po nastavení bodov prerušenia spustíme program voľbou Run-Run (Ctrl-F9). Začne sa vykonávať normálne, ale akonáhle narazí na bod prerušenia, zastaví sa a v hlavnom okne IDE je zobrazený zdro-



Obr. 2

jový kód s pásom vykonávania na riadku s bodom prerušenia. (Pamätajme si, že pokiaľ pás vykonávania (modrý) prekrýva riadok s bodom prerušenia, zvýraznený pás bodu prerušenia (červený) nemožno vidieť.)

Ak teraz do okna Watch vložíme akékoľvek premenné alebo výrazy, zobrazia sa ich aktuálne hodnoty.

V takto pozastavenom programe na bode prerušenia môžeme použiť niekoľko spôsobov ladenia:

– môžeme krokovať program voľbami Trace Into, Step over, Go to cursor vrátane testovania a modifikácie premenných
– môžeme pridávať alebo rušiť výrazy v okne Watch
– môžeme nastaviť alebo zrušiť bod prerušenia
– môžeme si prezrieť výstup programu použitím Window-User screen (Alt-F5)
– môžeme program prerušiť a znovu od začiatku spustiť Run-Program Reset a Run-Run
– môžeme pokračovať v behu programu po ďalší bod prerušenia voľbou Run-Run.

K odstráneniu bodu prerušenia presunieme kurzor na príslušný riadok a stlačíme opäť Ctrl-F8. Tento príkaz funguje ako prepínač, teda zapne alebo vypne bod prerušenia.

Keď ladíme veľmi veľký program, obzvlášť ak je zložený z viacerých unitov, môže sa nám stratiť náväznosť (čo vlastne chceme robiť). Turbo Pascal má vstavanú navigáciu. Pre jej podporu debugger poskytuje dva príkazy: **Window-Call stack** a **Search-Find Procedure**.

Call Stack

Po každom volaní procedúry alebo funkcie si TP toto volanie vrátane počtu predaných parametrov pamätá, pretože si informáciu uloží do zásobníku. Po ukončení podprogramu je volanie zo zásobníku uvolnené a vykonávanie je vrátené volajúcemu podprogramu. Kedykoľvek, keď sa program zastaví v mieste prerušenia alebo pri krokovaní, môžeme požadovať stav volania zásobníku príkazom Window-Call stack (Ctrl-F3). Objaví sa okno a zobrazí zoznam aktuálnych volaní podprogramov práve aktívnych v zásobníku (obr. 2). Môžeme si spätne prezrieť sekvenciu volania. To, čo sa do zásobníku uloží ako prvé, to bude zobrazené v najvrchnejšej časti sekvencie. Po zozname volania v zásobníku sa môžeme pohybovať šípkami. Keď stlačíme medzerník, prenesieme sa na posledný aktívny bod v programe. Volanie zásobníku zaznamená až 128 vnorených volaní.

Vyhľadávanie procedúr alebo funkcií

Počas ladenia je niekedy potrebné vyhľadať určitý podprogram k nastaveniu bodu prerušenia, ku kontrole zoznamu parametrov a pod.

Ak máme zdrojový kód rozdelený do viacerých súborov alebo unitov, určite voľbu Search-Find Procedure uvítame. Voľba vyvolá malé okno, kde vložíme identifikátor a stlačíme Enter (obr. 3). Potom pascal skontroluje svoje vnútorné tabuľky, procedúru vyhľadá a do okna zavedie zdrojový kód s kurzorom nastaveným na začiatok podprogramu.

Teraz sa pýtate, na čo je to dobré. No, až vytvoríte skutočne rozsiahly projekt o desaťtisícich riadkoch v 29 unitoch, potom mi uveríte!

Ako písať programy pre ladenie?

Vieme, že každý krok ladenia vykoná všetky príkazy na jednom riadku. Preto spravidla píšeme zdrojové texty tak, že čo riadok, to jeden príkaz. Ak však vieme, že niektoré priradenia hodnôt premenným sú bezchybné, môžeme ich napísať na jeden ria-

dok aj viac. (Pozor! Riadok nesmie byť väčší ako 127 znakov!).

Ale vôbec, najlepšie ladenie je preventívne ladenie. Dobre odladený a napísaný program bude mať nielen menej chýb, ale bude jednoduchšie ho sledovať a vyhľadávať chyby, ak sa vyskytnú.

Pri písaní veľkých programov sa snažme držať týchto zásad:
- pokiaľ je to možné, kódujeme, testujeme a ladíme program po malých častiach. Preveríme každú časť, než ju pridáme k ďalšej časti
- rozdelíme program do modulov: unitov, procedúr, funkcií. Vyhneme sa (nie vždy sa to dá!) písaniu podprogramu väčšieho ako 25 riadkov, kvôli prehľadnejšiemu prezeraniu na obrazovke
- predávame informácie v rámci programu cez parametre, miesto odkazov na globálne premenné vo vnútri procedúr. Vyhneme sa tak stranovým efektom a vytvoríme modul, ktorý ma zreteľne definované vstupy a výstupy.

Otázky pamäti

Počas ladenia veľkého programu môže chýbať pamäť. Turbo Pascal obsahuje súčasne v pamäti počítača editor, kompilátor, debugger, aktuálny zdrojový text, vykonateľný kód, symbolické tabuľky a iné ladiace informácie. Množstvo voľnej pamäte môžeme sledovať príkazom File-Get Info. Je mnoho spôsobov, ako získať viac voľnej pamäte a ich popis by zabral samostatný článok. Môžeme ich rozdeliť na dve stránky – mimo (vonkajšiu) alebo vo vnútri (vnútornú) prostredia IDE. Do vonkajšej stránky patrí vhodné nastavenie súboru CONFIG.SYS a AUTOEXEC.BAT. Vyradíme z nich nepotrebné rezidentné programy, zmeníme nastavenie BUFFERS a FILES a pod. Menej skúsení v tejto oblasti nech radšej požiadajú o pomoc veci znalého odborníka, lebo by inak mohli zapríčiniť haváriu systému. Vnútornú stránku prostredia IDE zvádnu aj tí menej zdatní. Veľmi pomáhajú tieto dve veci:
– nastaviť voľbu **Compile-Destination** na *Disk*
– nastaviť voľbu **Option-Linker-Linker buffer** na *Disk*

Keby ani toto nepomáhalo, raďte sa pokynami z helpu.

Keď sme si objasnili spôsoby ladenia programu, bolo by vhodné, aby sme si povedali,

Kde nemožno ladiť

V niektorých prípadoch nemožno trasovať alebo krokovať vo vnútri podprogramu. Je to obvyklé (ale nie vždy), pretože zdrojový text procedúry, ktorý chceme ladiť, nie je dostupný. Takéto situácie sú nasledujúce:
– akákoľvek procedúra alebo funkcia typu inline. Dôvod je ten, že podprogram **inline** nie je funkčným alebo procedurálnym volaním. Na miesto volania (call) je vkladaný príslušný strojový jazyk a takéto volanie sa správa ako jednoduchý príkaz
– akýkoľvek podprogram TP zo štandartných unitov (Crt, Dos, Graph, Graph3, Overlay, Printer, System, Turbo3)
– akákoľvek procedúra alebo funkcia typu **external**
– akákoľvek procedúra alebo funkcia typu **interrupt**
– akákoľvek procedúra alebo funkcia alebo inicializačný kód, ktorý nebol kompilovaný s direktívou {$D+} alebo so zapnutou voľbou **Option-Compiler-Debug Information**
– akákoľvek procedúra alebo funkcia alebo kód v unite, ktorého zdrojový text nemožno nájsť. Ak dostanete od niekoho unit v preloženom tvare .TPU, kde poznáte iba jeho funkcie a procedúry, nebudete ich môcť ladiť trasovaním F7, ale len krokovaním ako jeden príkaz F8.
– akákoľvek procedúra nastavená ako procedúra ukončenia. Pri trasovaní F7 sa nikdy nedostanete do Exit procedúry. Je tu však možné nastaviť bod prerušenia.

Tááák, máme za sebou teoretické, ale veľmi dôležité intermezzo. Sami zistíte, že aj keď teraz ešte nepoužívate všetky naznačené možnosti, programátorina vás k tomu časom donúti. Ale nabudúce sa už zase budeme venovať programovaniu.

STRETNUTIE S PASCALOM

9. časť

Úvod do druhého cyklu

Máme za sebou prvý cyklus programovania. Vysvetlili sme si čo to zo základov tvorby programu, jeho ošetrenia a položili sme dôraz na prítulnosť (user-friendly) programu k užívateľovi. Po krátkom intermezze o spôsoboch odladovania sa v tomto cykle budeme venovať tomu, ako programovo sprístupniť tzv. systémové súčasti počítača, ako je čas, disky, súbory, klávesnica, obrazovka, tlačiareň a iné „fajnovosti“. V dnešnej časti si preberieme

Hranie s časom

Nie, nie, ak si niekto myslí, že pomocou počítača zmeníme čas (myslím ten ozajstný, ľudský), tak nech obráti list. My sa budeme hrať s časom a dátumom počítača. Tí skorej narodení si isto ešte pamätajú na prvé ixtéčká, ktoré nemali vstavané hodiny a tak sa po každom spustení počítača musel dátum a čas nastavovať ručne. Dnes už však chvalabohu všetky mainboardy hodiny majú, a tak táto únavná práca odpadá.

Čas (myslím tým hodiny aj dátum) je v PC zabezpečovaný špeciálnym integrovaným obvodom, ktorý je napájaný z malej baterky, spravidla naletovanej na základnej doske. To zabezpečuje, že po vypnutí napájacieho napätia do PC sa nastavenie času nielen nestratí, ale hodiny a dátum bežia ďalej. Po opätovnom zapnutí počítača sa vyčíta tento čas z tohto obvodu a počítač a príslušné programy s ním pracujú normálne. Čas sa používa v počítači veľmi často, napr. pri ukladaní súborov, spúšťaní určitých, na čase závislých programov a pod.

Nastavovanie času

Čas môžeme nastavovať v počítači tromi spôsobmi: v SETUPe, v DOSe a programovo.

V SETUPe sa hodiny a dátum nastavujú spravidla pri zostavovaní počítača a jeho súčastí, kedy v počítači nie je ešte žiadny operačný systém ani iné programy. Vzhľadom na veľký počet úkonov je tento spôsob nastavenia v bežnom živote nevhodný.

V DOSe (je jedno v ktorom: MS-DOS, PC DOS, DR DOS, Novell DOS) sa na nastavenie času používajú dva príkazy: DATE a TIME. Ako už z názvu vyznieva, DATE slúži na nastavenie dátumu a TIME na nastavenie hodín. Pre ilustráciu si vyskúšajte obidva príkazy.

Programové nastavovanie času

Ak však chceme využívať alebo nastavovať dátum a hodiny v našich programoch, musíme k integrovanému obvodu pristúpiť programovo. Turbo Pascal používa na prácu s časom tieto štyri základné procedúry:
– GetDate
– SetDate
– GetTime
– SetTime

GetDate(var rok, mesiac, deň, deň v týždni : word) vracia aktuálne nastavený dátum v operačnom systéme počítača. Hodnoty: rok 1980..2099, mesiac 1..12, deň 1..31, deň v týždni 0..6, kde nula zodpovedá nedeli.

SetDate(var rok, mesiac, deň : word) nastaví aktuálny dátum do operačného systému. Ak dátum nie je v platnom rozsahu, požiadavka na nastavenie bude ignorovaná.

GetTime(var hodina, minúta, sekunda, stotiny sekúnd : word) vracia aktuálne hodiny nastavené v OS. Rozsah vrátených hodnôt je: hodina 0..23, minúta 0..59, sekunda 0..59, stotiny sekúnd 0..99.

SetTime(var hodina, minúta, sekunda, stotiny sekúnd : word) nastavuje aktuálne hodiny do OS. Ak hodnoty nie sú v platnom rozsahu, je požiadavka na nastavenie ignorovaná.

Práca s hodinami

Ako ilustračný príklad procedúr GetTime a SetTime je program **HODINY.PAS** na listingu č. 1. V hlavnom tele programu sa nachádzajú nám už známe príkazy, takže ich nebudeme podrobne rozoberať. Spomeňme si, že existuje mnoho iných a efektívnejších spôsobov vyčitávania stlačených klávesov, ale tu som použil najjednoduchší spôsob. Prvá procedúra vráti stav hodín z operačného systému a uloží do premenných hod, min, sek, stot typu word. Tento stav sa vypíše na obrazovku a ponúkne sa možnosť zmeniť ho.

Ak stlačíme malé "a", tak si program vypýta nové hodnoty hodín a minút a potom procedúrou SetTime tieto hodnoty nastaví do OS. Vypíše nové nastavenie hodín a ukončí sa.

Ak sme zadali malé "n", tak sa na obrazovku vypíše hlásenie „Pôvodný čas“ a program sa ukončí.

Pozrime sa bližšie na špeciálnu funkciu

Vloz_Nulu(w : word) : string.

Táto funkcia skontroluje veľkosť vstupného čísla w a prevedie ho na reťazec. Ak je číslo w dvojmiestne, prevedie ho do dvojmiestneho reťazca (napr. 17). Ak je však číslo w jednomiestne (teda v rozsahu 0..9), pridá do výstupného reťazca nulu, aby sa toto číslo zobrazovalo ako dvojmiestne (napr. 00, 03 a pod). Vieme, že hodiny sa takto zobrazujú a my predsa dodržíme dohodnutú konvenciu!

Listing. č. 1:

```
program Hodiny;

uses Dos;

var
  hod, min, sek, stot : Word;
  co : char;
  pom : word;

function Vloz_Nulu(w : Word) : String;      {ak je cislo jednomiestne}
var                                         {tak vlozi pred neho nulu}
  s : String;
begin
  Str(w:0,s);
  if Length(s) = 1 then
    s := '0' + s;
  Vloz_Nulu := s;
end;

BEGIN
  GetTime(hod,min,sek,stot);                {vrat cas}
  WriteLn('Je prave ',Vloz_Nulu(hod),':',  {vypis}
          Vloz_Nulu(min),':',Vloz_Nulu(sek),
          '.',Vloz_Nulu(stot));
  Writeln('Chces zmenit cas? (a/n)');
  readln(co);
  if co = 'a' then
    begin
      Writeln('Zadaj hodinu');
      Readln(pom);
      hod:=pom;
      Writeln('Zadaj minuty');
      Readln(pom);
      min:=pom;
      SetTime(hod,min,sek,stot);
      Writeln('Nastaveny cas je ',Vloz_Nulu(hod),':',
        Vloz_Nulu(min),':',Vloz_Nulu(sek),'.',Vloz_Nulu(stot));
    end
    else Writeln('Povodny cas');
END.
```

Práca s dátumom

Hlavné telo programu **DATUM.PAS** na listingu č. 2 je veľmi podobné predchádzajúcemu príkladu. GetTime vráti aktuálny dátum v premenných y, m, d, dow typu word. Funkcia Den_v_poradi je veľmi zaujímavá. Ako vstupy tejto funkcie sú hodnoty dňa a mesiaca, ako výstup je číslo, ktoré udáva, koľký deň daného mesiaca je to od 1. januára v bežnom roku.

Listing. č. 2:

```
program Datum;

uses dos, crt;

const
  days : array [0..6] of String[7] =          {tabulka dni}
    ('Nedela','Pondelok','Utorok',
     'Streda','Stvrtok','Piatok',
     'Sobota');

var y,m,d,dow,pom : word;
    x : integer;
    co : char;

function Den_v_poradi(den,mes :word) : integer;  {urcenie poradia}
var p1 : word;
begin
   case mes of                  {ak je mesiac cislo}
     1 : p1:= 0;                {tak sucet dni do zaciatku mesiaca}
     2 : p1:= 31;
     3 : p1:= 31+28;
     4 : p1:= 59+31;
     5 : p1:= 80+30;
     6 : p1:= 120+31;
     7 : p1:= 151+30;
     8 : p1:= 181+31;
     9 : p1:= 212+31;
    10 : p1:= 243+30;
    11 : p1:= 273+31;
    12 : p1:= 304+30;
   end;
   Den_v_poradi:=p1+den;    {plus pocet dni v tomto mesiaci}
end;

BEGIN
   ClrScr;
   GetDate(y,m,d,dow);          {vrat datum}
   x:=Den_v_poradi(d,m);        {vypocet poradia}
   writeln('Dnes je ',d,'.',m,'.',y,' ',days[dow],' a je to ',x,'. den v roku');

  Writeln('Chces zmenit datum? (a/n)');
  readln(co);
  if co = 'a' then
    begin
      Writeln('Zadaj den');
      Readln(pom);
      d:=pom;
      Writeln('Zadaj mesiac');
      Readln(pom);
      m:=pom;
      Writeln('Zadaj rok');
      Readln(pom);
      y:=pom;
      SetDate(y,m,d);
      Writeln('Nastaveny datum je ',d,'-',m,'-',y);
    end
     else Writeln('Povodny datum');
END.
```

Princíp algoritmu:
Ak zadáme dátum 29. 3. tak do premennej p1 sa uloží číslo 59, teda súčet dní januára (31) a februára (28). K tomuto súčtu sa pripočíta počet dní v stanovenom mesiaci (teda 29). Výsledok je výstup funkcie, teda v tomto prípade je to 31+28+29=88. Takto vypočítame, že 29. marec je 88. deň v roku. Teraz vás asi napadla otázka, ako to bude, keď je priestupný rok, ako tento. No áno, toto musíme ošetriť a tak to dostávate za domácu úlohu!

A načo je dobré vedieť, koľkýže to je deň v poradí? No to veľmi využijeme v

Programe LETO.PAS.

Videli ste už Windows'95? Majú jednu veľmi zaujímavú funkciu: vedia automaticky prestaviť hodiny pri prechode na letný čas a naspäť. Algoritmus tohoto výpočtu je pomerne zložitý, pretože Windows'95 uvažujú toto pre rôzne roky. A že je táto funkcia teraz už ochromená, asi už viete. Európsky parlament rozhodol,

že návrat z letného času k slnečnému bude o mesiac neskôr. (Ten Billy musí mať asi radosť!!!).

My si podobnú funkciu teraz naprogramujeme. Pre jednoduchosť si to nebudeme komplikovať rokmi, ale vieme, že v tomto roku letný čas začína 31. marca o 2,00 hodine ráno (hodiny si posunieme o jednu hodinu vpred) a končí (podľa Európskeho parlamentu) 27. októbra o 3,00 hod ráno (hodiny si posunieme o jednu hodinu späť).

Ako to bude fungovať?

Program LETO by mal byť prítomný pri každom spustení počítača, najlepšie v AUTOEXECu. Po spustení program vyčíta zo systému dátum a hodiny. Zistí, či vyčítaný čas je v rozmedzí letného času, teda či patrí do intervalu 31. 3. 02,00 hod až 27. 10. 03,00 hod. Potom z pomocného konfiguračného súboru LETO.CFG zistí nastavenie premennej stav. Premenná stav určuje, či už program LETO letný čas nastavil alebo nie. Je to veľmi dôležité, inak by program posúval hodiny s každým zapnutím počítača, a preto si informáciu o prvom nastavení uložíme do premennej stav v súbore LETO.CFG! Táto dosahuje len dve hodnoty: "a" – už bol nastavený, "n" – ešte nebol nastavený letný čas (bol vypnutý).

Na základe týchto dvoch hodnôt (vyčítaný čas systému a stav) má program 4 možnosti:

– vyčítaný čas je letným časom a ešte nebol nastavený (31. 3. – 27. 10 a stav = "n")

Ak nastane táto možnosť, program LETO posunie hodiny o jednu dopredu a do konfiguračného súboru zapíše "a", že už čas nastavil.

– vyčítaný čas je letným časom a už bol nastavený (31. 3. – 27. 10. a stav = "a")

Ak nastane táto možnosť, program nič nezmení, lebo vie, že už bol letný čas raz nastavený.

– vyčítaný čas nie je letným časom ale letný čas bol nastavený (1. 1. – 30. 3. alebo 28. 10. – 31. 12. a stav = "a")

Program zistí, že doba letného času pominula, ale ten bol nastavený, takže posunie hodiny o jednu späť a do konfiguračného súboru zapíše "n", že vypol letný čas.

– vyčítaný čas nie je letným časom a letný čas už bol vypnutý (1. 1. – 30. 3. alebo 28. 10. – 31. 12. a stav = "n")

Program zistí, že nie je letný čas a ten už bol vypnutý, takže nič nemení

Podľa týchto štyroch možností sa bude program správať.

Pozrime sa podrobnejšie na listing č. 3 programu LETO.PAS.

V deklaračnej časti sa okrem premenných nachádzajú už známe funkcie Den_v_poradi a Vloz_Nulu. Funkcia Je_letny_cas je typu boolean a dosahuje dvoch hodnôt – true alebo false, podľa toho, či vyčítaný čas funkciami GetDate a GetTime je v rozsahu 31. 3. 02,00 hod. až 27. 10. 03,00 hod. alebo nie. Toto zistíme takto: vyočítame poradie dnešného dňa a uložíme do premennej dnes. Do premennej teraz uložíme počet hodín, ktorý ubehol od 1. januára 00,00 hodín do spustenia programu. Podobným spôsobom zistíme počet hodín do 31.marca do 02,00 hod (let_hod) a takisto do 27. októbra do 03,00 hodín (zim_hod). Ak je vyčítaný čas v rozmedzí týchto hodín, nadobúda funkcia Je_letny_cas hodnotu true, ak nie, tak false. Asi sa pýtate, prečo to kontrolujeme na hodiny. No preto, že čas sa mení o druhej, respektíve o tretej hodine ráno a nemôžeme si byť istí či niekto nepracuje aj v týchto hodinách.

Listing. č. 3:

```
program Leto;

uses Dos,crt;

var
  y, m, d, dow,hod,min,sek,stot  : Word;
  dnes,letny_den,zimny_den : integer;
  teraz,let_hod,zim_hod    : longint;
  subor : text;
  stav  : char;

function Den_v_poradi(den,mes :word) : integer;
var pl : word;
begin
   case mes of
     1 : pl:= 0;
     2 : pl:= 31;
     3 : pl:= 31+28;
     4 : pl:= 59+31;
     5 : pl:= 80+30;
     6 : pl:= 120+31;
     7 : pl:= 151+30;
     8 : pl:= 181+31;
     9 : pl:= 212+31;
    10 : pl:= 243+30;
    11 : pl:= 273+31;
    12 : pl:= 304+30;
   end;
   Den_v_poradi:=pl+den;
end;

function Je_letny_cas : boolean;
begin
  Je_letny_cas := false;
  dnes:=Den_v_poradi(d,m);
  teraz:=(dnes-1)*24 + hod;
  letny_den:=Den_v_poradi(31,3);
  let_hod:=(letny_den-1)*24 + 2;
  zimny_den:=Den_v_poradi(27,10);
  zim_hod:=(zimny_den-1)*24 + 3;
  if (teraz >=let_hod) and (teraz<= zim_hod) then
Je_letny_cas := true;
end;

function Vloz_Nulu(w : Word) : String;
var
  s : String;
begin
  Str(w:0,s);
  if Length(s) = 1 then
    s := '0' + s;
  Vloz_Nulu := s;
end;

procedure Nastav;
begin
  if (Je_letny_cas) and (stav='n') then     {prva moznost}
     begin
       hod:=hod+1;
       SetTime(hod,min,sek,stot);
       stav:='a';
       Rewrite(subor);
       Writeln(subor,stav);
       Close(subor);
       Writeln('Zistil som prechod na letny cas');
       Writeln('Nastavil som cas
',Vloz_Nulu(hod),':',Vloz_Nulu(min));
     end;

  if (Je_letny_cas) and (stav='a') then     {druha moznost}
     begin
       WriteLn('Dnes je ', d:0,'.',m:0,'.');
       Write('Cas je ', Vloz_Nulu(hod),':',Vloz_Nulu(min));
       if stav='a' then writeln(' letneho casu.')
         else write(' SEC');
     end;

  if not(Je_letny_cas) and (stav='a') then  {tretia moznost}
     begin
       hod:=hod-1;
       SetTime(hod,min,sek,stot);
       stav:='n';
       Rewrite(subor);
       Writeln(subor,stav);
       Close(subor);
       Writeln('Zistil som prechod na slnecny cas');
  Writeln('Nastavil som cas ',Vloz_Nulu(hod),':',Vloz_Nulu(min));
     end;

  if not(Je_letny_cas) and (stav='n') then   {stvrta moznost}
     begin
       WriteLn('Dnes je ', d:0,'.',m:0,'.');
```

```
      Write('Cas je ', Vloz_Nulu(hod),':',Vloz_Nulu(min));
      if stav='a' then writeln(' letneho casu.')
        else write(' SEC');
    end;

end;

BEGIN                                   {vymaz obrazovky}
  Clrscr;
  assign(subor,'leto.cfg');              {otvorenie suboru}
  Reset(subor);                         {precitanie premmenej}
  Readln(subor,stav);
  Close(subor);                         {zatvorenie suboru}
  GetDate(y,m,d,dow);                      {vycitanie datumu}
  GetTime(hod,min,sek,stot);                {vycitanie casu}
  Nastav;                              { a rozhodni!}
END.
```

Jedinná procedúra nášho programu Nastav rieši vyššie preberané štyri možnosti činnosti programu. Iba sme ich rozšírili o hlásenie o stave, aby sme vedeli pravú skutočnosť.

V hlavnom tele programu vymažeme obrazovku, otvoríme konfiguračný súbor, zavrieme ho, vyčítame zo systému dátum a hodiny. Na základe týchto zistených premenných v procedúre Nastav vykoná program stanovenú činnosť a ukončí sa.

Ešte si musíme povedať niečo o konfiguračnom súbore LETO. CFG. Je to obyčajný textový súbor o jenom znaku, ktorý vytvoríme v ľubovoľnom textovom editore okrem T602 (tá vkladá na začiatok súboru svoje špecifické znaky). Nazveme ho LETO. CFG a zapíšeme do neho znak "a" alebo "n". Či to bude "a" alebo "n", to záleží, či program konfigurujeme v letnom alebo slnečnom čase. Pozor, uvedomme si, čo nastavujeme!

Precvičili sme si prácu s časom. Dokonca náš program nie je iba cvičný, ale je aj užitočný. Ak ho dopracujete do „user-friendly“ podoby, môžete ho šíriť do sveta ako shareware alebo freeware, veď kto by nechcel mať zautomatizovaný prechod na letný čas a späť?

Nabudúce sa pohráme s tlačiarňou.

STRETNUTIE S PASCALOM

10. časť

Vieme, že každý solídny program, ktorý spracováva určité údaje, má výstup na tlačiareň. Dnes už sú cenové relácie tlačiarní také, že je možné si kúpiť tlačiareň aj domov bez toho, aby sme zruinovali rodinný rozpočet. A tak si teraz povieme niečo o tom, ako je možné z programového hľadiska tlačiareň ovládať tak, aby výstupy našich programov boli nielen úžitkové, ale mali aj určitý vzhľad a komfort.

Rozdelenie typov tlačiarní a princíp ich činnosti som vysvetľoval v seriáli „Slovenčina v počítači“ v starších číslach tohoto časopisu, a preto sa s tým teraz nebudeme zdržiavať. My sa dnes budeme venovať stále najdostupnejším ihličkovým tlačiarniam. Ako vzor si vyberieme tú, ktorá už podruhýkrát vyhrala súťaž 10 NAJ, a tou je STAR LC 20. Ale keďže patrí medzi najrozšírenejšie štandarty, programy písané pre ňu budú vo väčšine prípadov fungovať aj na iných značkách tlačiarní bez potrebných úprav. Pozrime sa, čo najviac potrebuje programátor ovládať prostredníctvom svojho programu na tlačiarni. Je to

– status – stav tlačiarne
– reset tlačiarne
– margins – okraje na papieri, medzi ktorými bude prebiehať tlač
– paper detect – zapínať alebo vypínať kontrolu konca papiera
– prepínať medzi znakovými sadami, či bude znak vybratý z pamäte ROM alebo RAM
– meniť font a typ písma
– meniť riadkovanie

Preberme si jednotlivé možnosti podrobnejšie:
Hádam najdôležitejším problémom pri programovom ovládaní tlačiarne je poznať

Status tlačiarne

Môžeme ho získať použitím funkcie BIOSu každého počítača. Použijeme prerušenie – interrupt 17, službu 02 – Get Printer Status. Ako sa táto služba používa, je vidieť na listingu č.1. Do registra AH vložíme číslo služby (teda 2), do registra DX vložíme číslo tlačiarne (0-LPT1, 1-LPT2, až 3, v našom prípade 0) a zavoláme prerušenie 17. Použitie tejto služby vráti v registri AH stav (status) testovanej tlačiarne. Status je veľký 1 byte, ktorého osem bitov určuje skutočný stav tlačiarne (pozri tabuľku č. 1). Aby sme mohli v programoch pracovať s jednotlivými bitmi jednoduchšie, vytvoríme si jednorozmerné pole o ôsmych položkách, ktoré nazveme Acc8. Toto pole naplníme hodnotami 0 alebo 1, kde jednotka značí, že daná položka je aktívna. Ak teda testom zistíme, že je v bite č. 5 jednotka, znamená to, že v tlačiarni nie je papier. Ak toto zapracujeme do nášho programu, môžeme veľmi efektívne obsluhe programu oznámiť, aby dala papier do tlačiarne. Obdobným spôsobom môžeme otestovať, či je tlačiareň v ON LINE alebo nie, túto chybu programovo ošetriť a vyzvať obsluhu k náprave. Takto sa stanú naše programy user-friendly. A o to nám ide. Vytvoríme procedúru

Status(var status_byte : Acc8);

ktorej výstupným parametrom je práve pole o veľkosti 8 bitov. Spôsob otestovania jednotlivých položiek status bajtu ve znaázornený v programe TEST.PAS na listingu č.2.

Na programové riadenie ďalších vymenovaných možností budeme používať tzv. Escape sekvencie tej-ktorej tlačiarne. Sú to riadiace znaky tlačiarne, ktoré začínajú znakom escape (Esc), po ktorom nasledujú riadiace pokyny. Na základe escape znaku tlačiareň vie, že nasledujú znaky nemá vytlačiť na papier, ale tieto nastavujú jej možnosti dané výrobcom. Escape sekvencie sú určitým svetovým štandartom, a preto sú (až na malé výnimky) u každej tlačiarne rovnaké. Bližšie sa o nich dočítate v manuáli svojej tlačiarne.

Druhou často potrebnou funkciou je

Reset tlačiarne

Reset zabezpečí, že sa tlačiareň znovuinicializuje ako pri vypnutí a zapnutí sieťového vypínača, teda načíta sa stav DIP prepínačov a pod. Dôležitým rozdielom pri resete je, že sa nevymaže pamäť RAM tlačiarne, a tak nami nadefinovaný download zostane zachovaný (pozri „Slovenčina v počítači“).

Escape sekvencia resetu tlačiarne je <ESC> "@". Ako ale vyšleme tieto znaky do tlačiarne v Turbo Pascale? Nadefinujeme si konštantu Esc = #$1B (pozri ASCII tabuľku,znak 27 dekadicky) a procedúru Reset_Printer. Jej definícia je v listingu č. 1.

Občas potrebujeme nastaviť okraje tlače inak, ako sú prednastavené pri zapnutí alebo resete tlačiarne. Použijeme na to funkcie <ESC> "l" n pre nastavenie ľavého okraja a <ESC> "Q" n pre nastavenie pravého okraja tlače. Číslo n je v rozpätí 0 až 255 a udáva počet znakových pozícií toho-ktorého okraja. Pre nastavovanie okrajov si vytvoríme procedúru

Nastav_Okraje(lavy,pravy:byte).

Často chceme tlačiť tesne nad dolný okraj papiera, napríklad pri vpisovaní do predtlačeného formulára alebo šeku. Mnohé tlačiarne majú snímač konca papiera vysoko, a tak tlačiareň neumožní tlač na toto miesto. Je síce možné prepínačom v tlačiarni vypnúť kontrolu konca papiera, ale to chce zakaždým tlačiareň inicializovať a stále prepínať DIP – prepínače (niektorí silní jedinci snímač papiera prelepujú leukoplastom, fuj!!!). My to budeme riešiť pomocou escape sekvencií a procedúr Vypni_kontr_pap a Zapni_kontr_pap.

Podobne budeme prepínať medzi výberom znakov z tabuľky v ROM alebo v RAM pamäti tlačiarne. V RAM môžeme mať nadefinovanú diakritiku a z pamäte ROM budeme využívať semigrafiku. Použijeme procedúry Nastav_RAM a Nastav_ROM.

Asi najčastejšie budeme prepínať medzi rôznymi fontami a typmi písma. Fonty sú Pica, Elite, Condens, Proportional, normal, kurzíva, široké, vysoké, podtrhnuté a mnohé ďalšie podľa možností tlačiarne. Typy sú DRAFT a NLQ pre 9-ihličkové tlačiarne, DRAFT, LQ pre 24-ihličkové tlačiarne. Pri využívaní je vždy nutné preštudovať manuál tlačiarne, či dané fonty a typy písma umožňuje. Na listingu je naznačený spôsob prepínania písma procedúrami Nastav_Pica, Nastav_Elite, Zapni_NLQ a Zapni_Draft, ostatné si naprogramujte sami. Nezabudnime, že štandartne (default) je v každej ihličkovej tlačiarni prednastavené pri zapnutí alebo resete PICA (10 znakov/palec) a Draft.

Ak chceme, aby určitý string (reťazec znakov) bol vypísaný napr. kurzívou, vytvoríme si procedúru Vytlac_Kurzivou(s:string), ktorá prepne tlačiareň do kurzívy, vytlačí spomínaný reťazec a znovu vypne kurzívu. Potom v hlavnom tele programu stačí volať túto procedúru namiesto neustáleho zapínania a vypínania fontu tlačiarne.

Pri vpisovaní do predtlačených formulárov, ako sú rôzne doklady, šeky, dokumenty a iné, je veľmi dôležité meniť aj riadkovanie pri tlači. Jednu kolonku môžeme tlačiť pri riadkovaní 1/8 palca, druhú pri 1/6 palca a tretiu napr. pri 15/72" (palca). Palec – inch = 2,54 mm. Štandartné riadkovanie je 1/6". Nastavenie programu na vpisovanie do konkrétneho formulára je veľmi náročné, vyžaduje presnosť, mnoho cvičných formulárov a určitú skúsenosť (a pevné nervy!). Odmenou je zautomatizovanie vypisovania týchto formulárov a tým aj značné uľahčenie tejto protivnej práce. Na to poslúžia procedúry Riadkovanie_8, Riadkovanie_6 a Riadkovanie_n_72.

Aby sme tieto vyššie naznačené možnosti mohli efektívne využívať v našich programoch, najlepšie bude, ak si napíšeme vlastný unit MyPrn. V časti INTERFACE nadefinujeme potrebné konštanty a premenné a vypíšeme názvy procedúr. V časti IMPLEMENTATION nadeklarované procedúry zostrojíme. Nezabudnime na inicializáciu, aj keď prázdnu (listing č.1).

Na listingu č.2 je cvičný program TEST.PAS, ktorý demonštruje činnosť unitu. Bližší popis je zahrnutý priamo v hlavnom tele programu ako reťazce prcedúry writeln.

Za domácu úlohu si doprogramujte procedúry a funkcie, o ktorých si myslíte, že ich budete pri práci s tlačiarňou potrebovať. A nabudúce sa pohráme s disketovými mechanikami, harddiskom, adresármi a súbormi.

Tabuľka č.1 Význam jednotlivých bitov status bajtu.

Bit	Váha	Význam anglicky	Preklad
0	01h	timeout	vypršanie časového limitu
1	02h	not used	nepoužité
2	04	not used	nepoužité
3	08	I/O error	vstupne-výstupná chyba
4	10h	selected	tlačiareň je vybraná
5	20h	out of paper	v tlačiarni nie je papier
6	40h	acknowledgement	potvrdenie-odpoveď tlačiarne
7	80h	not busy	nie je zaneprázdnená

Listing č. 1:

```
unit MyPrn;

INTERFACE

uses crt, dos, printer;

const
    Esc      = #$1B;
    FF       = #$0C;
    CR       = #$0D;
    LF       = #$0A;

type  Acc8= array[0..7] of 0..1; {stavový bajt}

var   reg : registers;
     ii  : integer;
     a   : Acc8;
     b,i : Byte;

procedure Status(var status_byte:Acc8);
{skontroluje stav tlačiarne a v prípade potreby informuje
užívateľa}

procedure Reset_Printer;
{znovu inicializuje tlačiareň}

procedure Nastav_Okraje(lavy,pravy : byte);
{nastaví okraje na tlače}

procedure Vypni_Kontr_Pap;
{vypne kontrolu konca papiera}

procedure Zapni_Kontr_Pap;
{zapne kontrolu konca papiera}

procedure Nastav_ROM;
{volí použitie znakov z ROM pamäte tlačiarne}

procedure Nastav_RAM;
{volí použitie znakov z RAM pamäte tlačiarne}

procedure Nastav_Pica;
{nastaví ¨!rku p!sma 10 zn./palec}

procedure Nastav_Elite;
{nastav! šírku písma 12 zn./palec}

procedure Zapni_Draft;
{tlač bude v type Draft}

procedure Zapni_NLQ;
{tlač bude v type NLQ}

procedure Riadkovanie_8;
{nastaví riadkovanie na 1/8"}

procedure Riadkovanie_6;
{nastaví riadkovanie na 1/6"}

procedure Riadkovanie_n_72(n:byte);
{nastaví riadkovanie na n/72"}

procedure Vytlac_Kurzivou(s:string);
{vytlačí iba zadaný reťazec kurzívou a potom kurzívu vypne}

IMPLEMENTATION

procedure Status(var status_byte:Acc8);
begin
  reg.DX:=0;          {LPT1}
  reg.AH:=2;          {služba Get Printer Status}
  intr($17,reg);      {volanie prerušenia 17}
  for i:=0 to 7 do
    begin
      status_byte[i]:=reg.AH mod 2; {napĺňame stavový bajt
ako pole}
      reg.AH:=reg.AH div 2;
    end;
end;

procedure Reset_Printer;
begin
  write(lst,esc,'@');
end;

procedure Nastav_Okraje(lavy, pravy :byte);
begin
  write(lst,esc,'l',chr(lavy));
  write(lst,esc,'Q',chr(pravy));
end;

procedure Vypni_kontr_pap;
begin
  write(lst,esc,'8');
end;

procedure Zapni_kontr_pap;
begin
  write(lst,esc,'9');
end;

procedure Nastav_ROM;
begin
  write(lst,esc,'%','0');
end; {ROM}

procedure Nastav_RAM;
begin
```

```
  write(lst,esc,'%','1');
end; {RAM}

procedure Nastav_Pica;
begin
  write(lst,esc,'P');
end;

procedure Nastav_Elite;
begin
  write(lst,esc,'M');
end;

procedure Zapni_Draft;
begin
  write(lst,esc,'x','0');
end;

procedure Zapni_NLQ;
begin
  write(lst,esc,'x','1');
end;

procedure Riadkovanie_8;
begin
  write(lst,esc,'0');
end;

procedure Riadkovanie_6;
begin
  write(lst,esc,'2');

end;

procedure Riadkovanie_n_72(n:byte);
begin
   write(lst,esc,'A',chr(n));
end;

procedure Vytlac_Kurzivou(s:string);
begin
  write(lst,esc,'4');
  writeln(lst,s);
  write(lst,esc,'5');
end;

BEGIN
END.
```

Listing č. 2:

```
program Test;

uses crt,myprn,printer;

var pole : Acc8;
   veta : string;

procedure Hlavicka_Statusu;
begin
  writeln(lst,'7 6 5 4 3 2 1 0');
end;

procedure Pata_Statusu;
begin
  Writeln(lst,          timeout');
  Writeln(lst,          not used');
  Writeln(lst,'         I/O error');
  Writeln(lst,          selected');
  Writeln(lst,         out of paper');
  Writeln(lst,'        acknowledgement');
  Writeln(lst,'       not busy');
end;

BEGIN
 Status(pole);
 if pole[3]=1 then begin
                writeln('Vstupne-výstupná chyba !',^g);
                writeln('ZAPNI DO ON LINE !!!',^g);
                writeln;
                writeln('Stlač ľubovoľnú klávesu !',^g);
                repeat until keypressed;
                Reset_Printer;
                halt(1);
              end;
 if pole[4]=1 then writeln('ON LINE ');
 if pole[5]=1 then
    begin
      writeln('Tlaciaren nema papier');
      Halt(1);
    end;
 Hlavicka_Statusu;
 for ii:=7 downto 0 do
   begin
     write(lst,pole[ii],' ');
   end;
 writeln(lst);
 Pata_Statusu;

 Reset_Printer;
 writeln(lst,'Toto je test fungovania statusu');
 Nastav_Okraje(20,60);
 writeln(lst,'Toto je test okrajov');
 Nastav_Elite;
 writeln(lst,'Toto je test pisma ELITE');
 Nastav_Pica;
 writeln(lst,'Toto je test pisma PICA');
 Zapni_NLQ;
 writeln(lst,'Toto je test pisma NLQ');
 Zapni_Draft;
 writeln(lst,'Toto je test pisma DRAFT');
 Riadkovanie_6;
 for i:=1 to 5 do writeln(lst,'Toto je test riadkovania 1/6"');
 Riadkovanie_n_72(15);
 for i:=1 to 5 do writeln(lst,'Toto je test riadkovania 15/72"');
 Riadkovanie_n_72(5);
 for i:=1 to 5 do writeln(lst,'Toto je test riadkovania 5/72"');
 Riadkovanie_8;
 for i:=1 to 5 do writeln(lst,'Toto je test riadkovania 1/8"');
 Nastav_Okraje(0,80);
 writeln(lst,'Zrusime okraje');

 Vytlac_Kurzivou('Toto je test kurzivy');

END.
```

STRETNUTIE S PASCALOM

11. časť

Viete, čo je to vlastne skratka DOS? No predsa Disk Operating System, teda diskový operačný systém. Ale prečo diskový? Lebo disky, či už floppy alebo hard sú snáď najdôležitejšou pripojitelnou súčasťou počítača. Bez diskov by nebolo možné programy spúšťať, ani po nich vyžadovať výstupné data a podobne. Stálí opozičníci namietnú, že v dobe sietí nám nebude treba diskov, existujú predsa bezdiskové stanice... Chyba, páni, zabudli ste, že disk je na serveri, a to poriadny! Takže počítač bez diskov je ako auto ...taktiež bez diskov.

Preto hlavné jadro každého operačného systému a BIOS-u je postavené na práci s diskami pomocou funkcií, ktoré sa nazývajú služby. Tieto služby sú očíslované a každé číslo reprezentuje jednú jedinečnú službu. Každá takáto služba má vstup (aspoň jeden – a to číslo vstupu) a výstup – hodnoty, ktoré chceme touto službou získať. Výstupných parametrov môže byť aj viac, to záleží na službe. Keďže sa jedná o veľmi dôležitú a náročnú oblasť, rozdelíme si prácu s diskami na dve časti: služby DOS-u a služby BIOS-u. V tejto časti si ukážeme prvé tri funkcie DOS-u.

Aby sme veľmi a dlho neteoretizovali, vytvoríme jednoduchý program, na ktorom si ukážeme možnosti ovládania diskov pomocou služieb DOS-u.

Už sme asi všetci videli program, ktorý automaticky zistí, aké mechaniky a disky sú k počítači pripojené, a akého sú typu. Že ste ešte taký program nevideli? Stačí sa pozrieť na hociktoré windowsy!

Náš program **DISKY** nebude taký pekný ako Windows, ale tiež dokáže zistiť počet diskových mechaník a určiť ich typ.

Vieme, že na počítači môže byť nainštalovaných viac diskov, napr. A, B, C, D..., kde vieme, že A a B sú floppy mechaniky a C a D sú harddisky. Tieto môžu byť vlastné, ale môžu to byť aj disky „pritiahnuté“ zo siete, označené nenasledujúcim písmenom, napr. K, L, atď.

V našom programe použijeme tri služby (podslužby) DOS-u:
– služba 19h – Zisti vybraný disk
– služba 44h a jej podslužba 08 – kontrola vymenitelnosti
09 – kontrola miestny/vzdialený

Ako sa vlastne pracuje s týmito funkciami?
Aby služba dala žiadané výsledky, musíme splniť tieto pravidlá: (nové slová, vzťahujúce sa k činnosti so službami sú vytlačené *kurzívou* – pochopte ich význam a používajte ich!)
– najprv naplníme registre procesora potrebnými vstupnými dátami napr. `r.Al:= $19; r.AX:=$4409; r.BL:=1;`
– službu zavoláme v Turbo Pascale takto: `MSDOS(r);`
– otestujeme výstupné parametre a podľa nich vetvíme činnosť programu `if r.Al <> $0F then...`

Tabuľka č. 1:

služba	podsl.	vstup	výstup
19h	–	AH=19h (číslo služby)	AL=číslo disku, 0=A,1=B..
44h	08	BL=číslo disku 0 – vybraný, 1-A, 2-B..	AX=00h – vymeniteľný 01h – nie je vymen. 0Fh – chyba!
44h	09h	Bl=číslo disku	DX=atribut zariadenia. Ak je 12.bit =1 tak je to disk v sieti.

Pozrime sa teraz na listing programu DISKY:PAS. Sledujte komentáre priamo na listingu. Po deklarácií premenných deklarujeme dve dôležité funkcie – function Drivers : string a funkction Typ_Disku(Disk:char):byte;.

Prvá funkcia *Drivers* vracia reťazec veľkých písmen, ktoré reprezentujú označenia diskov v počítači, na ktorom sa tento program spúšťa. Ako funguje?

Naplníme register AH číslom služby (19hex). Uvedenýcm príkladom Turbo Pascalu službu zavoláme. V registri AL sa vráti hodnota predstavujúca aktuálny disk takto: 0=A, 1=B, 2=C... a podobne. V cykle od A po Z najprv skúsime nastaviť disk príkazom *ChDir*. Ak *IOResult* vráti nulu, znamená to, že disk naozaj existuje, a tak sa jeho písmenové označenie vloží do premennej *zoznam*. Ak nie, neuloží sa nič. Ďalším príkazom kontrolujeme, ktorý disk je posledný. teraz využijeme vrátenú hodnotu registra AL. Príkazom *Ch(r.AL+65)* prevedieme číselné označenie disku na písmenové a posledným *ChDir* sa vrátime na disk, z ktorého sme vychádzali. Výstupom funkcie *Drivers* je reťazec existujúcich znakov.

Druhá funkcia *Typ_Disku(Disk:char):byte* má vstupný parameter písmenové označenie disku a vracia byteovú číselnú hodnotu, ktorá charakterizuje typ doski takto:
0 = disk neexistuje
1 = disketová floppy machanika (vymeniteľný)
2 = hard disk
3 = sieťový (vzdialený) disk

Činnosť funkcie je takáto: Register BL naplníme číselným označením disku podľa tabuľky. Register AX naplníme číslom služby a podslužby 4409hex. (Znak $ v Turbo Pascale symbolizuje hexadecimálne čísla.) Zavoláme službu a testom na registri *Flags, FCarry* a DH skontrolujeme, či je disk miestny alebo vzdialený, teda sieťový.

Znovu naplníme register AX službou a podslužbou 4408hex a službu zavoláme. Návratová hodnota v registri AL určí vymeniteľnosť kontrolovanej mechaniky.

V hlavnom tele programu sú už známe príkazy na výmaz obrazovky a výpis hlásení. Premennú *mech* naplníme obsahom funkcie *Drivers*. testom od A po posledný disk v počítači (teda nemusí to byť „Z“!) otestujeme každý dostupný disk a vypíšeme jeho typ. jednoduché, nie?

Ak postavíme testy vo funkcii *Typ_Disku* prísnejšie, prípadne pridáme inú službičku DOS-u alebo BIOS-u, môžeme napr. určiť, či mechanika A je 3,5" a má kapacitu 1,44 MB, alebo zistiť, či jeden z diskov nie je náhodou CD-ROM. Potom môžeme v našich programoch vytvoriť nádherné ikony, ake maju u Billa Gatesa. Môžeme na to použiť službu... ale urobte si niečo aj sami!

Nabudúce budeme pokračovať v hre s diskami pomocou BIOS-u a ukážeme si aj pár veľmi pekných trikov, ktoré prekvapia vašich priateľov, Nie, nie, nebojte sa, len nijaké výrusy.

Listing č. 1:

```
program Disky; {zisti disky v PC a urci ich typ}

uses dos,crt;

var
    mech   : string;
    a      : byte;
    z      : char;
    posledny: char;

function Drivers:string;      {zisti dostupnost diskov}

var
    n    : char;
    r    : registers;
    zoznam: string[26];

begin
  r.ah:=$19;      {sluzba 19hex DOSu}
  MsDos(r);       {volanie tejto sluzby}
  zoznam:='';
  {$I-}
  for n:='A' to 'Z' do      {od A po Z}
    begin
      ChDir(n+':');      {nastav disk}
      if IOResult=0 then  {a otestuj, ci naozaj existuje!!!}
        begin
          zoznam :=zoznam+n; {ak ano, zarad ho do zoznamu}
           posledny:=n;      {posledny disk je zapamatany}
        end;
    end;
  {$I+}
  ChDir(Chr(r.al+65)+':');
  drivers:=zoznam;              {vystup f-cie je zoznam pismen}
end;{Drivers}                   {diskov, dostupnych v PC}

function Typ_Disku(Disk:char):byte;  {zisti typ disku}

var r:registers;

begin
  Typ_Disku:=0;
   r.BL:=ord(Disk)-64; {zmena oznacenia z pismena na cislo A=1,B=2...}
  r.AX:=$4409;       {sluzba DOSu 44, podsluzba 9 hex}
  MsDos(r);          {volanie sluzby}
  if (r.Flags and FCarry)=FCarry then Exit;
  if (r.DH and 16)=16 then Typ_Disku:=3          {test typu - sietovy?}
    else
      begin
        r.AX:=$4408;  {sluzba 44, podsluzba 8 hex}
        MsDos(r);     {volanie}
       if r.Al <> $0F then Typ_Disku :=r.al+1; {test typu - vymenitelny?}
      end;
end;{Typ_Disku}

BEGIN
  clrscr;
  mech:=drivers;
  writeln('Disky sú :',mech);
  for z:='A' to posledny do
   begin
     a:=Typ_Disku(z);
     case a of
```

```
        0 : writeln(z,' neexistuje !');
        1 : writeln(z,' je typu Floppy.');
        2 : writeln(z,' je typu Hard.');
        3 : writeln(z,' je sietovy disk.');
      end;
    end;
END.
```

STRETNUTIE S PASCALOM

12. časť

Je horúce leto, komu by sa chcelo študovať a programovať, a preto si dnes povieme niečo o produkte, ktorý programovanie v Pascale robí omnoho jednoduchším a hlavne efektívnejším. Týmto zároveň odpovedám na mnohé vaše listy, kde ma žiadate o získanie alebo doporučenie určitých knižníc k TP.

Vieme, že niektoré často používané procedúry a funkcie môžeme uložiť do tzv. unitov, čo je niečo ako knižnica týchto rutín. Potom stačí v hlavnom programe danú rutinu iba zavolať, čo je jednoduchšie ako ju znova naprogramovať. V predchádzajúcich častiach seriálu sme vytvorili vlastné knižnice, síce veľmi jednoduché, ale funkčné. Isto ste si aj sami napísali iné, rozsiahlejšie unity. Ja mám mnoho takých knižníc, ktoré som napísal veľmi dávno, používam ich, ale už ani neviem, ako som ich vytvoril. Verte či neverte, časom sa to stane aj vám...

Načo by sme ale pracne programovali unity, ktoré už niekto elegantne napísal? Mnoho takýchto unitov sa dá získať ako shareware na rôznych BBS-kách alebo najnovšie na CD ROM-och. Autor ich dodáva buď v preloženom tvare (.TPU) a k tomu interfejsovú časť alebo priamo aj so zdrojovým textom. Mám skúsenosť, že niektoré knižnice by som asi napísal lepšie, ale bolo viac takých, ktoré niesli v sebe krásnu funkciu alebo boli poriadne zložité a bolo vidieť, že programátor (alebo tím programátorov) odviedol kus dobrej roboty. Najlepšie je, keď niekde získame unit aj so zdrojákom. To potom môžeme niektoré funkcie preprogramovať (pozor na autorské práva), ale hlavne každý zdroják je výborný študijný materiál. Žiadna učebnica ma nenaučila toľko, čo funkčný zdrojový text. (Aj z niektorých nefunkčných zdrojákov – sú aj také – sa dá vyčítať dobrá myšlienka, keď už nič, tak ako sa to robiť nemá.

Čím ďalej, tým viac je možné nájsť knižnice v jazyku slovenskom alebo českom. To potom aj programy už komunikujú v týchto jazykoch.

Ešte o tých jazykoch: angličtinu mám rád, hlavne tú technickú. Neviem si predstaviť preklady čisto anglotechnických výrazov, ako sa o to niekto násilne presadzuje. (Videl som interface = medzistyk, no nech, ale čo taký handshake?) Na druhej strane ak robím program pre niekoho, robím to vždy v jemu zrozumitelnom jazyku.

Ako zákon schválnosti káže, keď si pracne urobím unit, za niekoľko dní podobný objavím ako shareware na niektorom disku alebo BBS-ke.

Avšak shareware spravidla rieši iba jednu oblasť v programovaní, napr. databázy, grafiku, komunikácie cez porty alebo modem, matematické výpočty, diskové operácie a pod. Ak chceme knižnice, zahrňujúce širokú „parketu“, musíme sa obrátiť na profesionálnu firmu. Preto si dnes povieme niečo o produkte brnenskej firmy TurboConsult. Jej produkt sa volá Friendly Pascal.

Výrobok mi poskytla firma vo verzii 2.5. Osobne som mal možnosť pracovať dlho so staršou verziou 2.0 a preto ich budem obidve porovnávať.

Charakteristika

Friendly Pascal je súbor nadstavbových knižníc – unitov pre Turbo Pascal. Poskytuje prostriedky pre vysoko efektívnu tvorbu aplikačných programov. Friendly Pascal zbavuje programátorov nutnosti zaoberať sa riešením obecných problémov, ale unožňuje plne sa venovať úlohám špecifickým pre vytváraný program.

Fiendly Pascal nie je len púhym balíkom užitočných procedúr a funkcií. Vzájomnou previazanosťou a jednotným štýlom riešenia svojich častí stelesňuje určitú programátorskú filozofiu, ktorá ovplyvňuje nielen vonkajší vzhľad vytváraných programov, ale aj ich vnútornú stavbu. Friendly Pascal poskytuje programátorom priateľské prostredie pre ich prácu a umožňuje im vyvíjať programy priateľské k užívateľom. Stadiaľ aj ten názov.

Dodávka

Friendly Pascalu (ďalej FP) obsahuje vo verzii 2.0 tri diskety 5.25". Na dvoch je samotný FP. Na tretej diskete sa nachádza demonštračný program, ktorý nám veľmi pekne naznačí všetky najdôležitejšie možnosti FP. K tomu manuál o 590! stranách v dnes už málo videnej pevnej väzbe. Nedá mi nezastaviť sa u väzby manuálu. Akoby v TurboConsulte tušili, že programátor bude s knihou často pracovať. Poznáte aj vy knižky v mäkkej väzbe, ktoré sa po jednom prečítaní hneď rozletia? Tak tu môžete (ba čo, musíte!) listovať veľmi často, ale listy držia a držia.

Verzia FP 2.5 je dodávaná na dvoch disketách 3.5" v peknom plastikovom obale. Ďalej tu nájdeme už spomínanú knižku v pevnej väzbe k verzii 2.0 a zalaminovanú knižku o 179 stranách k verzii 2.5. Vzhľadom k tomu, že verzia 2.5 prináša zmeny oproti 2.0 len okrajovo alebo vôbec, to nevadí. Podstatné zmeny, zvlášť koblasti grafiky, ktoré verzia 2.0 neriešila, sú popísané v druhej tenšej knižke. Avšak na disketách už nenájdete demonštračný program z verzie 2.0, čo je na škodu veci. Možnosti FP v súhrnej podobe teda neuvidíte. Naopak, zatiaľ čo FP 2.0 sa dodával len v preloženom tvare (mimo niektorých dôležitých unitov) a kto chcel zdrojové texty, musel ich doplatiť, verzia FP 2.5 obsahuje zdrojové texty (či chcete alebo nechcete) všetky.

Hardwarové požiadavky

Verzia 2.0 sa uspokojila s Turbo Pascalom 5.5 alebo 6.0, MS-DOS 3.0 a vyššie a pracoval od XT-čiek a viac. FP 2.5 je trošku náročnejší: požaduje MS DOS 3.3 a viac, Tubo Pascal 7.0 (len pre programovanie v chránenom režime, inak stačí aj TP 6.0) a procesor 286 a viac a najmenej 2 MB pamäti. Keďže počítam, že dnes je to najzákladnejšie minimum, čo má programátor v dosahu, netreba si robiť zbytočné starosti. Vzhľadom k tomu, že FP podporuje myš, bolo by dobré ju pripojiť, ale nie je nutnosťou.

Inštalácia

FP je v obidvoch prípadoch jednoduchá. Zabezpečuje ju inštalačný program v štýle Borlandu, a navádza programátora postupne krok za krokom. Cieľ inštalácie sa môže meniť podľa potreby. FP 2.5 obsahuje na druhej diskete program na generovanie českých fontov, a preto je dobré si prečítať príslušné pokyny v súboroch readme.

Tu by som mal povedať, že FP nenahrádza Turbo Pascal. FP je nadstavbou nad Borlandovským Turbo Pascalom, plne využíva jeho IDE prostredie a hlavne kompilátor. Ale svojou skladbou nevyužíva borlandovské knižnice v plnej miere (v pravom zmysle slova), ale svoje. Takže ak chcete používať Friendly Pascal, musíte mať aj Borlandov TP.

Po inštalácii je nutné dopísať v TP cestu k novým unitom a potom už nič nebráni plnému využitiu Friendly Pascalu.

Použitie FP unitov je klasické. Stačí v klauzuli uses dopísať ten-ktorý názov unitu a môžeme použiť datové štruktúry a funkcie tohoto modulu.

Príklady

FP obsahuje aj zdrojové texty príkladov ku každej kapitole alebo oblasti. Niektoré sú len niekoľko riadkové a predvádzajú možnosti napr. ovládania tlačiarne alebo tvorby menu, iné komplexnejšie riešia prácu s dátami v širokom meradle od zadávania cez editáciu a filtráciu až po tlač výstupných zostáv. Tieto prí-

klady slúžia nielen na ukážku možností využitia jednotlivých procedúr a funkcií, ale hlavne určitou modifikáciou vzorového príkladu je aj priemerný programátor schopný vytvoriť kvalitný úžitkový program na profesionálnej úrovni. No a ako som spomenul, sú zdrojom študijných podkladov.

Skladba a charakteristika modulov Friendly Pascalu

FP tvorí okolo dvoch desiatok programových modulov – unitov, ktoré poskytujú prostriedky pre riešenie problémov a vyskytujú sa priamo alebo v istej obmene pri tvorbe väčšiny aplikačných programov. Sú označované ako FP moduly a môžeme ich charakterizovať takto:

FPTools – je podporný program celeho systému. Pracuje s reťazcami a netypovými premennými. Obsahuje deklarácie datových štruktúr, používaných v ostatných FP - moduloch

FPDate – definuje premenné a základné operácie s časom a dátumom, ako sú zmeny hodnôt, prevody do textového tvaru a naopak atď.

FPCrt – poskytuje ucelený súbor prostriedkov pre ovládanie obrazovky. Využíva princíp virtuálnych obrazoviek ako tzv. listov. Týchto môže byť na obrazovke aj niekoľko naraz a ich počet a veľkosť sú obmedzené len veľkosťou pamäte. Výstupné operácie môžu byť smerované do ľubovolného listu, pričom sa listy môžu prekrývať. FPCrt umožňuje automatickú detekciu použitého adaptéru (EGA,VGA,Hercules) a podľa toho môže nastaviť používanie národných znakov. Súčasťou modulu sú prostriedky pre obsluhu myši, klávesnice a zvukového generátoru.

FPCrtUse – je aplikáciou FPCrt pri riešení často používaných siuácií. Umožňuje rôzne možnosti výstupu na obrazovku, vytváranie rámčekov, rastrov a iné.

FPMenu – je originálny prostriedok na rýchlu tvorbu menu typu „pull-down“, ako napr. v T602. Podporuje ovládanie z klávesnice alebo myšou.

FPEdit – je vysoko účinný aparát pre zadávanie dát pomocou formulárov, vytvorený prostriedkami objektovo orientovaného programovania (tzv. OOP. Tí čo už o tom niečo viete, prisvedčíte mi, že je to základná filozofia programovania vo Windows. Ale o tom hádam inokedy). Umožňuje pomocou aparátu virtuálnych obrazoviek vytvárať a editovať formuláre väčšie ako skutočná obrazovka a automaticky zabezpečuje rolovanie obsahu. Rieši dynamickú zmenu štruktúry formulára v závislosti na hodnotách zadávaných z klávesnice (to tie objekty!).

FPEdUse – je konkrétnym použitím modulu FPEdit.

FPCommun – je ukážkou riešenia bežných situácií pri komunikácii obsluhy s programom: oznámenie chyby, informácie o prevádzanej činnosti, výpisy správ pre obsluhy napr. o výmene papiera v tlačiarni a pod.

FPAccess/FPAccsL – rieši problematiku prístupu k datovým súborom. Poskytuje prostriedky pre napĺňanie súborov dátami, prístup k dátam, automaticky kontroluje integritu súborov. Vychádza z koncepcie modulu Toolbox firmy Borland. Výborný prostriedok na tvorbu programov databázového charakteru, napr. kartotéky osôb a iné.

FPSort/FPSortL – umožňuje triedenie dát podľa určitých kritérií.

FPLst – obsahuje prostriedky pre obsluhu tlačiarne a tvorbu tlačových zostáv ako je detekcia chybových stavov, nastavovanie tlačovej pozície, stránkovanie a iné. Rieši tlač hlavičiek a predpätia stránky alebo prerušenie tlače obsluhou. Dopĺňa to, čo úplne chýba v klasickom Turbo Pascale.

FPLib – poskytuje aparát pre obsluhu súboru so špeciálnou štruktúrou, ako sú katalógy, číselníky a pod.

FPHelp – každý zložitejší program má v sebe kontextovú nápovedu k určitej činnosti programu. Tento modul umožňuje jednoducho vytvárať takýto help.

FPNation – sústreďuje všetky prostriedky FP-modulov, ktoré môžu meniť jazykovú variantu programu. Obsahuje definície konštant, textových hlásení v jazyku českom, slovenskom, nemeckom a anglickom, v abecede Kamenických alebo Latin 2, taktiež národné formáty výpisu dátumu a času. Ak raz vytvoríte program v slovenčine, pomocou kompilácie a menších úprav ho prevediete do iného jazyka pomerne rýchlo. Zabezpečuje triedenie podľa týchto abecied. (Ostatní tvorcovia programov, poučte sa!)

FPColors – poskytuje prostriedky pre nastavovanie atribútov menu, formulárov alebo dialógových boxov v závislosti na video adaptéri.

FPDOS – umožňuje aktiváciu dosovských služieb priamo z programu.

FPTrace – je jednoduchý prostriedok na odladovanie vytváraného programu.

Zmeny vo verzii FP 2.5

FPDos a FPSwap rieši prekonanie fyzickej hranice počítača. Zároveň v spolupráci s TP 7.0 je možné vytvárať aplikácie v chránemon režime.

Zmena v FPCrt podporuje prácu v grafickom režime obrazovky. Je možné využívať všetky procedúry pre prácu s grafikou, doplneného naviac o prostriedky pre prácu s formátom PCX.

Práce s textom boli vyčlenené z FPCrt a prepracované do nového modulu FPText.

Prostriedky grafického rozhrania obsahuje nový modul FPGraph, ktorý na modul FPCrt úzko nadväzuje. Tomuto modulu je zároveň venovaná druhá príručka vo verzii 2.5.

Dokumentácia

Okrem vyššie spomínanej väzby je možné hodnotiť dokumentáciu k produktu ako výbornú. Je písaná veľmi zrozumiteľne, všetky pasáže začínajú podrobnou charakteristikou premenných a filozofiou modulu. Tam, kde je to nutné, sú popísané konflikty s inými funkciami a prípadné chybové hlásenia a ich ošetrenie. Podstatnú časť dokumentácie zaberá presný popis jednotlivých procedúr a funkcií. Každá rutina je popísaná názvom, deklaráciou, parametrami. Popis je veľmi rozsiahly a podrobný, ukončený upozornením a možnými chybami. Záver každej rutiny obsahuje zoznam príbuzných funkcií. Spravidla každá kapitola je ukončená popisom a výpisom vzorového príkladu, ktorý sa nachádza aj na disku ako súbor.

Chybové stavy

FP vlastní svoj aparát na identifikáciu runtime-ových chýb. V manuáli sú tieto chybové hlásenia vysvetlené aj sprípadným riešením. Zo svojej praxe potvrdzujem, že mi tento aparát chým uľahčil odladovanie alebo prevádzku databázového programu. Za to vďaka, autori !

Nedostatky

Musím priznať, že mám voči FP jednu veľkú výhradu. Autori tohoto projektu zvolili dosť nešťastne ako základ práce s databázovými súbormi ToolBox od Borlandu. V našich končinách by sme asi radšej prijali štandart DBase, aby práca so súbormi typu .DBF bola jednoznačnejšia. Potom by bolo možné súbory využiť aj inak, prípadne ich navzájom zdieľať. Verím, že v ďalšej verzii toto autori uvedú na správnu mieru.

Friendly Pascal kontra Turbo Vision

Mám Borlandov rád. Ale myslím, že ich prídavok – Turbo Vision je nedotiahnutý. Borlandovci by sa mohli od brnenských Turbo-Consult-antov učiť, čo znamená „user-friendly“. (Dúfam, že sa neurazia, to by som nerád!)

Celkové hodnotenie

Friendly Pascal je výborný produkt. Na zapracovanie s ním je potrebný určitý čas, a neviem si predstaviť používanie jednotlivých modulov bez doprovodnej dokumentácie. Ten, kto sa s ním zžije, zistí, že tvorba ďalších programov je odbremenená od otrockej práce navrhovania užívateľského prostredia programu. Tá programová jednoduchosť a čistota, komplexnosť riešenia a hlavne zrozumiteľnosť (tá technická aj jazyková) predurčuje projekt k trvalému využitiu.

Ako základnú výhodu by som uviedol, že nie je nutné zvládnuť FP ako celok naraz. Po osvojení si určitej časti môže programátor naštudovať a využiť postupne ďalšie časti balíku. Ja som hľadal riešenie, ako jednoducho vyriešiť obsluhu tlačiarne a programu s užívateľom. Takto som objavil FP 2.0, kde som využil len modul FPLst. Postupne som pridal FPCommun, potom prečo nevyužiť aj FPDate a FPCrt. Keď som chcel využiť FPEdit, musel som si naštudovať objekty. Hovorím musel, ale teraz som za to vďačný! A nakoniec som zistil, že používam FP nie ako balík, ale skutočne ako filozofiu. Teraz to vyznie jednostranne, ale ja si iný kompletný produkt pre programovanie v DOS-e ani neviem predstaviť. O jazykovej výhode (snáď vám nevadí čeština) nehovoriac.

Vo FP je možné napísať jednoduchý program na prevod textu alebo kompletné účtovníctvo podniku. Vstavané ochranné prvky modulov zjednodušujú ošetrenia chybových stavov, takých bežných pri klasickom programovaní. A tak mi neostáva nič iné, len povedať, že program firmy TurboConsult Friendly Pascal je skutočne friendly.

Program Friendly Pascal 2.5 zapožičala k recenzii firma TurboConsult s. r. o., Botanická 56, 656 32 Brno, telefón 05/4121 2898, fax 05/41215603. Bohužiaľ mi ani na požiadanie neposkytla aktuálny cenník tohoto produktu. Ale keď im zavoláte, isto sa dohodnete.

A nabudúce budeme zase programovať.

STRETNUTIE S PASCALOM

13. časť

Dnes si preberieme funkcie BIOS-u pre prácu s diskami. Ako je u BIOS-u bežné, každej súčasti počítača sa venuje jedno prerušenie s jeho podriadenými službami. Operáciám s diskami je priradené prerušenie INT13h (dúfam, že už viete, že malé "h" za číslom znamená hexadecimálny tvar a 13h sa nerovná 13 dekadicky) a služby od nuly po 18h. My si vysvetlíme a vyskúšame iba najzaujímavejšie z nich.

Ako sa pracuje so službami BIOS-u? Podobne ako so službami DOS-u, teda najprv naplníme vstupné registre a to tak, že do registra AH vložíme číslo služby, do ostatných konkrétne parametre, aké tá ktorá služba vyžaduje, potom zavoláme prerušenie 13h (v TP intr), a v registroch služba vráti výstupné údaje. Podrobnosti o vstupe a výstupe sú v tab.č.1. Ideme na to:

00h Reset disku

inicializuje radič a mechaniky pružných alebo pevných diskov, vrátane rekalibrácie čítacej a záznamovej hlavičky vystavením na nultú stopu. Ktorú jednotku chceme resetovať nastavíme v registri DL takto: 00h = "A", 01h = "B", ale 80h = "C" atď...

Výstup tejto služby je stav operácie, uložený zase v registri AH a význam je v tab. č.2. Toto je veľmi potrebný údaj pri ladení programu a pri ošetrovaní chýb!

02h Čítanie z disku

načíta do pamäte (na ktorú ukazujú registre ES:BX) požadovaný počet sektorov z pružného alebo pevného disku. Adresa sektoru na disku sa zadáva číslom stopy, sektoru a hlavičky. Ak je v registri DL číslo menšie, ako je 80h, jedná sa o pružné – floppy mechaniky, ak je rovné, jedná sa o 1.harddisk, 81h je druhý HD atď.

Číslo stopy je 10-bitová hodnota (0 – 1023). Dva najvyššie bity (8,9) tejto hodnoty sa zapisujú do šiesteho a siedmeho bitu registra CL, ktorého dolných šesť (0-5) bitov obsahuje číslo počiatočného sektoru. (Niektoré BIOS-y podporujú až 12-bitové číslo stopy pre väčšie harddisky. Vtedy sa 10. a 11. bit zapisujú do najvyšších bitov registra DH). My však budeme zatiaľ skúšať iba na disketách, aby sme si „nepošahali“ harddisk, a tak nás tento na prvý (ale ozaj len prvý) pohľad zložitý odstavec nemusí trápiť.

03h Zápis na disk

zapisuje na požadovaný počet sektorov údaje z pamäti (ES:BX) na pevný alebo pružný disk. Adresa sektoru na disku sa zadáva číslom stopy, sektoru a hlavičky. Ostatné platí ako pri službe 02h.

Z konštrukčného hľadiska floppy mechaník môže pri operáciách s nimi dôjsť ku chybe z dôvodu ešte nízkych otáčok motora. Preto je nutné tieto operácie skúšať aspoň trikrát za sebou, a až potom hľadať chybu. Majme toto na pamäti a vyhneme sa zbytočným nepríjemnostiam.

05h Formátovanie stopy

formátuje všetky sektory na specifikovanej stope. Vstupné parametre sú popísané v tab. č.1. Pozrime sa bližšie na parameter ukazateľ na tabuľku formátu. Tento parameter sa používa iba pri formátovaní diskiet. Tabuľku tvorí zoznam adresných značiek všetkých sektorov, ktoré majú byť na danej stope formátované. Každú adresnú značku tvorí postupnosť štyroch bajtov. Napr. pre formátovanie 9 sektorov na stope (disketa 360 kB, 5,25") bude mať zoznam 9x4=36 bajtov. Adresná značka je definovaná takto:

1.bajt C – číslo stopy
2.bajt H – číslo hlavičky
3.bajt R – číslo sektoru
4.bajt N – veľkosť sektoru: 0 - 128 bajtov
1 – 256 bajtov
2 – 512 bajtov
3 – 1024 bajtov

Bežné diskety MS DOS-u sa formátujú na sektory veľké 512 bajtov. Zoznam adresných značiek pre 9 sektorovú disketu bude vyzerať takto:

CHRN CHRN CHRN ... CHRN
0112 0122 0132 ... 0192

08h Parametre disku

podávajú informácie o počte hlavičiek, počte stôp, počte sektorov na stope a počtu mechaník zadaného disku.

Nás bude teraz zaujímať iba výstupný parameter v registri BL pri práci s disketou. Ten nám povie informáciu, akú že to máme mechaniku (pozri tab.č.1). POZOR! Nesmieme si zmýliť typ disketovej mechaniky a diskety. Táto služba totiž parametre vyčitáva z pamäti CMOS, ale nekontroluje fyzické médium. Preto ak do mechaniky 1,2 MB 5,25" vložíme disketu o kapacite 360 kB, po zavolaní tejto služby zistíme, že máme mechaniku 1,2 MB a nie kapacitu diskety. Ak si toto neuvedomíme, môžeme dostávať skreslené výsledky. (Aby sme toto mohli korektne vyriešiť, musíme použiť aj inú službu BIOS-u, kde vyčítame veľkosť média.)

Ak som vám v predminulej časti seriálu naznačil, ako priradiť krásne ikonky mechanikám tak ako vo Windows, tak tu sa to začína...

15h Typ disku

hovorí o tom, či testovaná jednotka je pevný disk alebo disketa, a či podporuje detekciu výmeny diskety. Toto využijeme v službe

16h Detekcia výmeny diskety

kedy služba zistí, či od posledného prístupu k tej-ktorej mechanike došlo k otvoreniu dvierok a vybratiu diskety. Názov služby je zavádzajúci, pretože kontrola vybratia diskety sa prevádza mechanicky (verte mi a neotvárajte disketové mechaniky) a ak tam strčíme tú samú disketu, čo sme pred chvíľou vybrali, služba povie, že došlo k výmene diskety aj keď to nie je pravda. Došlo iba k jej vybratiu (a znovuvloženiu).

Zapáčili sa vám tieto služby? Dokážete si už predstaviť ich praktické použitie? Ale ako ich v Tubo Pascale použiť?

Aby sa nám to nezdalo také zložité, tak si to precvičíme.

Hádam najkrajšia a najefektívnejšia je služba 08h – Parametre disku. Na listingu č.1 je zdrojový text programu DISKETY.PAS. Veľmi jednoducho demoštruje použitie služby. Deklarujeme jedinnú premennú "r" typu registers a jednu parametrickú procedúru Testuj(typ : char). Register AH naplníme číslom služby a register DL číslom mechaniky v závislosti na označení, ktorú chceme testovať. Príkazom intr($13,r) službu voláme, teda chceme aby sa vykonala. Následným príkazom case testujeme výstup v registri BL a vypisujeme hlásenie. Chcem vás upozorniť na častú chybu: znak $ (dollar) pred číslom nahrádza označenie hex. Ak ho opominieme, nebudeme volať prerušenie 13 hex, ale 13 dekadicky, čo je 0Dhex a program isto nebude fungovať. Zvlášť si dajme pozor pri napĺňaní registrov, kedy po hodnotu 9 nemusíme znak dollar písať, lebo 9hex = 9dec. Hlavné telo programu sa skladá z vymazania obrazovky a z volania nadeklarovanej procedúry už s parametrom písmena mechaniky A aj B.

Tento program je veľmi jednoduchý a cvičný. Ale ak ho použijete ako procedúru vo svojom programe, pridáte ikony a poprípade máte grafický výstup, možno bude aj Bill závidieť...

Profesionálny program QMCOPY kopíroval diskety tak, že vôbec nepotreboval vloženie diskety potvrdiť stlačením klávesu. Jednoducho zistil, že som disketu už vložil. Ako? Myslím, že testoval vybratie a vloženie diskety do mechaniky pomocou služby 16h – detekcia výmeny média. Na listingu č.2 je zdrojový text programu VYMENA.PAS, ktorý demonštruje použitie tejto služby. Má už známu premennú r a pomocnú premennú *j* typu boolean. Deklarujeme funkciu Podporuje(typ : char) :boolean, ktorá má vstupný parameter písmeno označujúce mechaniku a sama je logická. Obsahuje volanie služby 15h – Typ disku, kedy zisťuje, či mechanika vôbec podporuje detekciu výmeny. Ak áno, nadobúda hodnotu true, inak nadobudne false. Deklarovaná procedúra Dvierka(typ : char) obsahuje test funkcie Podporuje a inak sa veľmi podobá na procedúru Testuj. Po jej zavolaní a otestovaní výstupných parametrov vypíšeme, či disketa bola z mechaniky vybratá alebo nie. Hlavné telo zase maže obrazovku a kontroluje disketu A aj B. Ak by sme sa chceli dopracovať k podobnému efektu ako v QMCOPY, stačí cyklicky testovať vybratie a vloženie diskety do mechaniky a nemusíme zaťažovať užívateľa stláčaním klavesu.

Azda najzložitejšie sa javí formátovanie, zápis a čítanie sektoru. Na listingu č.3 je zdrojový text programu SKUSKA.PAS, ktorý demonštruje použitie týchto funkcií. Je o trochu zložitejší ako predchádzajúce programy, ale zistíte, že obsahuje skoro tie isté procedúry. Predom si nadeklarujeme nový typ TBuffer, ktorý bude slúžiť ako odkladací priestor v pamäti. Je to sedemprvkové pole bajtov.

Deklarujeme pomocné premenné s,f,w,c typu bajt, premennú buffer typu TBuffer a niekoľko ďalších.

Prvou funkciou je WriteTrack, ktorá zapíše na 39.stopu nultej hlavy od prvého sektoru obsah v premennej Buffer. Toto opakuje maximálne trikrát, alebo pokým služba neprebehla správne. Počet opakovaní je závislý od premennej i (kvôli motoru). Obdobná je funkcia ReadTrack, ktorá prečíta daný sektor na danej stope a uloží jeho obsah do premennej Buffer. Funkcia ResetDisk je veľmi jednoduchá a nastavuje radič a hlavičky do východzej polohy. Vo funkcii FormatTrack je nadeklarovaná konštanta Tab typu TTab, čo je pole 36 prvkov o veľkosti bajtu na prvok. Je to nadefinovaná tabuľka adresných značiek presne podľa vyššie popísaného spôsobu. Každá z popísaných funkcií vracia jednobajtovú hodnotu obsahu registra AH, aby sme podľa tabuľky č. 2 mohli zistiť, kde nastala prípadná chyba. V hlavnom tele programu sa okrem zmazania obrazovky prevedie reset disku. Všimnime si, že služby môžu byť definované ako procedúry ale aj ako funkcie. Záleží len na nás. Vypíše sa stav resetu, aby sme vedeli, ako prebehol. Na obrazovke sa objaví jednoduché menu a načítaním klávesnice funkciou ReadKey a následným testom vetvíme program. Ak sme stlačili "z" ako zápis, program otvorí súbor s menom SKUSKA.INP a prečíta jeho obsah a uloží do premennej Buffer. Prevedie formátovanie stopy, vypíše kontrolu stavu operácie formátu a to, čo uložil do premennej Buffer zapíše na daný sektor. Vypíše stav priebehu operácie zápisu.Uzavrie súbor a program sa ukončí.

Tabuľka služieb BIOS-u pre prácu s diskami

Č.f.	Popis	Vstup	Výstup
00h	Reset disku	AH=0 DL=číslo jednotky 0h-A, 01h-B, 80h-C	AH=stav operácie CF=1 v prípade chyby
02h	Čítanie z disku	AH=02h AL=počet sektorov CL=počiat. sektor CH=číslo stopy DL=číslo jednotky DH=číslo hlavičky ES:BX=adresa pam.	AH=stav operácie AL=poč. načítaných sekt. CF=1 v prípade chyby
03h	Zápis na disk	AH=03h AL=počet sektorov CL=počiat. sektor CH=číslo stopy DL=číslo jednotky DH=číslo hlavičky ES:BX=adresa pam.	AH=stav operácie AL=poč. načítaných sekt. CF=1 v prípade chyby
05h	Formát stopy	AH=05h AL=počet sektorov CH=číslo stopy DL=číslo jednotky DH=číslo hlavičky ES:BX=ukazateľ na tabuľku formátu	AH=stav operácie CF=1 v prípade chyby
08h (pre floppy mech)	Parametre disku	AH=08h DL=číslo disku 1 – 360 kB, 5,25" 2 – 1,2 MB, 5,25" 3 – 720 kB, 3,5" 4 – 1,44 MB, 3,5"	AX=0 BL=0 – neznámy typ
15h – Typ disku		AH=15h DL=číslo jednotky	AH=typ disku: 0-nie je inštalovaná 1-floppy, nepodporuje detekciu výmeny 2-floppy, podporuje detekciu výmeny 3-pevný disk CF=1 v prípade chyby
16h	Detekcia výmeny diskety	AH=16h DL=číslo jednotky	AH=stav operácie: 0h-d. nebola vymenená 1h-chybné číslo mech. 6h-d. bola vymenená 80h-mech. nie je pripravená

Ak sme stlačili "c" ako čítanie, tak procedúra ReadTrack prečíta obsah daného sektoru a uloží v pamäti do premennej Buffer. Vypíše stav operácie čítania. Vytvorí a otvorí súbor s menom SKUSKA.OUT a zapíše do neho obsah premennej Buffer. Súbor uzavrie a ukončí sa. Aby sme mohli skúšať tento program, musíme si vytvoriť súbor SKUSKA.INP. Vytvoríme ho ľubovoľným editorom (nie v T602). Ja používam Norton Commander, stlačím Shift-F4, zadám do okienka meno a sám NC mi tento súbor ponúkne k editácii. Súbor naplníme znakmi, maximálne však siedmymi, inak by sme museli zmeniť číslicu 7 v procedúre BlockWrite a BlockRead. Ak program pracuje správne, tak obsah súboru SKUSKA.INP sa prenesie do súboru SKUSKA.OUT.

Nebojte sa skúšať! Ladťe programy, kontrolujte obsahy registrov v okne WATCH pri krokovaní, meňte obsahy registrov a premenných. Len tak si dobre osvojíte tieto služby a pochopíte ich činnosť.

Tabuľka chybových kódov prerušenia 13hex

kód	Popis chyby
00h	nedošlo k chybe
01h	zlý príkaz, zlá žiadosť pre radič
02h	zlá značka začiatku sektoru
03h	disketa chránená proti zápisu
04h	sektor s daným číslom nenájdený
05h	chyba pri resetovaní radiča AT
08h	chyba radiča DMA
09h	žiadosč na zápis za hranicu 64 kB (DMA)
10h	zlý CRC (parita)
40h	požadovaná stopa nenájdená
80h	jednotka nie je pripravená

A popritom vás isto napadnú perfektné aplikácie.

Nabudúce si ešte zodpovieme niekoľko problémových otázok a ukážeme zopár fínt, a potom sa zameriame na špecialitku – OOP – objektovo orientované programovanie. Hovoríte, že na čo je to dobré? Tak bez objektov, páni a dámy, v programovaní pod Windows ani krok! A tam (teraz) smeruje vývoj.

Listing č. 1:

```
program Diskety;

uses Dos, crt;

var
     r     : registers;

procedure Testuj(typ : char);
begin
   r.ah:=8;
   if typ = 'A' then  r.dl:=0; {disk. mechanika A};
   if typ = 'B' then  r.dl:=1; {disk. mechanika B};
   intr($13,r);
   write('Mechanika ',typ,' je typu ');
   case r.bl of
     0 : writeln('Neznamy typ');
     1 : writeln('360 kB,  5,25"');
     2 : writeln('1,2 MB,  5,25"');
     3 : writeln('720 kB,  3,5"');
     4 : writeln('1,44 MB, 3,5"');
   end;
end;

BEGIN
   Clrscr;
   Testuj('A');
   Testuj('B');
END.
```

Listing č. 2:

```
program Vymena;        {zistuje, ci bola disketa vymenena}

uses dos, crt;

var  r : registers;
     j : boolean;

function Podporuje(typ:char):boolean;
begin
   Podporuje:=false;
   r.ah:=$15;
   if typ = 'A' then  r.dl:=0; {disk. mechanika A};
   if typ = 'B' then  r.dl:=1; {disk. mechanika B};
   intr($13,r);
   if r.AH = 2 then Podporuje:=true
    else
   Podporuje:=false;
end;

procedure Dvierka(typ : char);
begin
   j:=Podporuje(typ);
   if not(j)  then
      begin
        writeln('Tento typ mechaniky ',typ,' nepodporuje detekciu
vymeny diskety.');
         exit;
      end
      else
   r.ah:=$16;
   if typ = 'A' then  r.dl:=0; {disk. mechanika A};
   if typ = 'B' then  r.dl:=1; {disk. mechanika B};
   intr($13,r);
   write('Disketa v mechanike ',typ,' ');
   case r.ah of
     0 : writeln('nebola vybrata');
     6 : writeln('bola vybrata');
   end;
end;

BEGIN
   Clrscr;
   Dvierka('A');
   Dvierka('B');
END.
```

Listing č. 3:

```
program Skuska;    {program na skusanie f-cii BIOSu}

uses dos,crt;

type
    TBuffer = array [1..7] of byte;     {urcenie typu bafra}

var
    s,f,w,c      : byte;
    buffer       : Tbuffer;             {bafr}
    subor,mysub  : file;                {vst-vystupne subory}
    result       : word;
    ch           : char;                {znak nacitany z klavesnice}

function WriteTrack(var buffer:TBuffer) : byte;
                                        {zapis sektoru}
var
    r : registers;
    i : byte;
begin
   i:=0;
   repeat
      r.ah:=3;  {zapis}
      r.al:=9;  {pocet sektorov}
      r.ch:=39; {cislo stopy}
      r.cl:=1;  {cislo pociatocneho sektoru}
      r.dh:=0;  {hlava}
      r.dl:=1;  {jednotka B=1;} {ak chcete mech.A, zmente na 0!}
      r.es:=Seg(Buffer);
      r.bx:=Ofs(Buffer);                {adresa buferu}
      Intr($13,r);
      inc(i);                           {volanie prerusenia}
   until (r.al=0) or (i>=3);            {opakuj max. trikrat}
   WriteTrack := r.ah;                  {hodnota f-cie}
end;

function ReadTrack (var buffer:TBuffer): byte;
                                        {citanie sektoru}
var
    r : registers;
    i : byte;
begin
   i:=0;
   repeat
      r.ah:=2;  {citanie}
      r.al:=9;  {pocet sektorov}
      r.ch:=39; {cislo stopy}
      r.cl:=1;  {cislo poc.sektoru}
      r.dh:=0;  {hlava}
      r.dl:=1;  {jednotka B=1;}
      r.es:=Seg(Buffer);
      r.bx:=Ofs(Buffer);                {adresa buferu}
      Intr($13,r);                      {volanie prerusenia}
      inc(i);
   until (r.al=0) or (i>=3);
   ReadTrack := r.ah;                   {hodnota f-cie}
end;

function ResetDisk :byte;               {reset disku}
var
    r : registers;
    i : byte;
begin
   i:=0;
   repeat
      r.ah:=0;  {reset}
```

```
    r.dl:=1;  {jednotka B=1;}
    Intr($13,r);                    {volanie prerusenia}
    inc(i);
  until (r.al=0) or (i>=3);
  ResetDisk:=r.ah;                  {hodnota f-cie}
end;

function FormatTrack:byte;          {formatovanie sektoru}

type TTab = array[1..36] of byte;

const Tab : TTab = (39,0,1,2,       {tabulka adresnych znaciek}
                    39,0,2,2,
                    39,0,3,2,
                    39,0,4,2,
                    39,0,5,2,
                    39,0,6,2,
                    39,0,7,2,
                    39,0,8,2,
                    39,0,9,2);
var
   r : registers;
   i : byte;

begin
  i:=0;
  repeat
    r.ah:=5;  {format}
    r.al:=9;  {pocet sektorov}
    r.ch:=39; {cislo stopy}
    r.dh:=0;  {hlava}
    r.dl:=1;  {jednotka B=1;}
    r.es:=Seg(Tab);
    r.bx:=Ofs(Tab); {adresa Tabulky}
    Intr($13,r);
    inc(i);
  until (r.al=0) or (i>=3);
  FormatTrack := r.ah;
end;

BEGIN
  clrscr;
  s:=ResetDisk;
  writeln('Stav operacie resetu : ',s);
  writeln('Z/Zapis');
  writeln('C/Citanie');
  writeln('Volba?');
  ch:=ReadKey;
  if ch = 'z' then
   begin
    assign(subor,'skuska.inp');
    reset(subor,1);
    BlockRead(subor,buffer,7);
    f:=FormatTrack;
    writeln('Stav operacie formatu : ',f);
    w:=WriteTrack(buffer);
    writeln('Stav operacie zapisu : ',w);
    close(subor);
   end;
  if ch = 'c' then
   begin
    c:=ReadTrack(buffer);
    writeln('Stav operacie citania : ',c);
    assign(mysub,'skuska.out');
    rewrite(mysub,1);
    BlockWrite(mysub, buffer,7);
    close(mysub);
   end;
END.
```

STRETNUTIE S PASCALOM

14. (záverečná) časť

Dnešnou časťou seriálu som chcel zahájiť objektovo orientované programovanie, ale na základe vašich listov a telefonátov som sa rozhodol venovať sa ešte klasickému štrukturovanému programovaniu. Dnes si preberieme veľmi dôležitú a veľmi často používanú súčasť programovania – prácu s databázovými súbormi formátu DBF. Keďže by som bol rád, aby v časopise vyšiel kompletný výpis zdrojových textov, aby ste si to mohli skúšať a hlavne používať, podstatne skrátim textovú časť článku. Dúfam, že to nebude na závadu, lebo ste už pokročilejší programátori a čo-to vám napovie aj komentár vo vlastných výpisoch.

Formát DBF zaviedla firma Ashton-Tate v programe pod známym názvom DBase III. Tento formát sa tak ujal, že ho prevzali aj ostatné firmy a používajú ho ako štandard a je viac-menej kompatibilný s programovými balíkami FoxPro a iné.

Nebudem rozpisovať konkrétnu štruktúru DBF súboru dopodrobna, v prípade potreby je možné toto naštudovať v dostupnej literatúre. Nám bude stačiť vysvetlenie základných pojmov a operácií s daným súborom.

Príklad databázovej tabuľky

	1.položka	2.položka	3.položka	felder.položka	
	MENO	PRIEZVISKO	NARODENY	ZENATY	PLAT
1.veta	Janko	Kukučka	29. 5. 1995	F	0
2.veta	Ferko	Kukučka	10. 2. 1910	T	28 000
3.veta	Anička	Ptáčková	30. 4. 1953	F	1470

recs-1.veta
recs.veta

DBF (ale aj hocijaký iný databázový) súbor si môžeme predstaviť ako tabuľku (obr. č. 1). Jednotlivé riadky tabuľky predstavujú vety súboru, stĺpce zase položky vety súboru. Obsah jednotlivých buniek sú data. Tabuľku budeme nazývať databáza (skrátene báza) a pre prácu programátora potrebujeme o nej vedieť tieto údaje:

– názov – je to meno súboru napr. OSOBY.DBF
– počet viet (niekedy nazývaných aj záznamy) – to je počet riadkov v danej tabuľke
– počet položiek (nazývaných aj pole) – to je počet stĺpcov danej tabuľky
– mená položiek – sú to názvy stĺpcov, píšu sa vždy VEĽKÝMI PÍSMENAMI, napr. MENO, ADRESA...
– typ a parametre položky – popisuje formát dát v položke. Sú to:
  reťazec – označuje sa "C", parameter udáva veľkosť v bajtoch,
  číslo – označuje sa "N", 1. parameter udáva veľkosť,
2. parameter udáva počet desatinných miest
  dátum – označuje sa "D" a obsahuje dátum
  logika – označuje sa "L" a dosahuje dvoch booleovských stavov: F – false a T – true.

Existujú aj ďalšie typy položiek (MEMO, BIN), ale tými sa zaoberať nebudeme.

Zatiaľ čo počet viet v databáze môžeme meniť (dopisovať, mazať, obnovovať), počet položiek vo vytvorenej databáze je nemenný. Sami prídete na to, že je možné určitými fintami už vytvorenú databázu reštruktualizovať (strašné slovo...), ale len na základe programových činností.

S databázami, vetami a položkami môžeme robiť tieto operácie:
– otvorenie databáze – sprístupnenie k ďalším operáciám
– zatvorenie databáze – korektné ukončenie činnosti, vyprázdnenie pracovných bufferov a premenných. V prípade, že by sme neukončili program uzavretím databázového súboru, môže dôjsť k strate dát a to by sme neradi.
– posun po vetách databáze – vpred, vzad, na začiatok, na koniec, skokovo
– vyčítanie dát, t.j. obsahu jednotlivých položiek konkrétnej vety
– naplnenie dát
– editácia a opravy dát položky
– výmaz vety
– obnova vety

Aby sme mohli v Turbo Pascale pracovať s databázovými súbormi typu DBF, musíme si napísať vlastný unit pre túto prácu, lebo TP nemá v sebe žiadnu funkciu na takéto operácie.

Na listingu č.1 je zdrojový text unitu DBASE3.PAS, ktorý zabezpečuje vyššie spomenuté základné operácie.

Hneď na začiatku zdrojového textu sú definície potrebných konštánt, typov a premenných, s ktorými budeme ďalej praco-

vať. Nasledujú deklarácie procedúr a funkcie pre prácu s DBF súbormi. V implementačnej časti sa nachádzajú definície jednotlivých procedúr a funkcie. Stačí len dôsledne opísať zdrojový text, uložiť s názvom DBASE3.PAS a preložiť. Dostaneme súbor DBASE3.TPU, ktorý používame už známym spôsobom zadefinovaním do klauzulky uses.

Použitie

jednotlivých procedúr je na vzorových príkladoch. Prvý, DBDEMO.PAS (listing č. 2) vytvorí databázový súbor TEST.DBF, ktorý má 3 vety o troch položkách, ktoré postupne napĺňame dátami. Obsah jednotlivých položiek je demonštračný a môžete ich ľubovoľne meniť.

Program DBLIST.PAS, ktorého zdrojový text je na listingu č. 3, demoštruje možnosti operácií s ľubovoľným DBF súborom. Meno súboru môže byť zadané na príkazovom riadku ako parameter alebo z klávesnice, keď si ho program vypýta. Použitím procedúry DBUse program zistí z hlavičky DBF počet viet v súbore (premenná recs), počet položiek vo vete (felder), mená položiek (feld[i].name), typ položky (feld[i].typ), veľkosť položky (feld[i].size) a v prípade numerického typu aj počet desatinných miest (feld[i].nk). V prvej časti vypíše štruktúru vety a zistené parametre, v druhej časti vypíše obsah jednotlivých položiek po vetách. V ďalšej časti skočí program na začiatok tabuľky (DBTop), teda na prvú vetu a vypíše obsah (data) vety. Potom skočí na koniec tabuľky (DBBottom), teda na poslednú vetu a vypíše jej obsah. Nakoniec sa posunie o jednu vetu späť (DBSkipBack), teda na predposlednú, lebo ukazateľ stál na poslednej pozícii a znova vypíše obsah vety.

Na základe týchto dvoch vzorových príkladov a listingu unitu ste schopní sami vypracovať praktické príklady na prácu s DBF súbormi. Aj keď sa jedná o veľmi jednoduchý unit s omedzenými možnosťami, plne vyhovuje pri používaní väčšiny DBF súborov a sám ho stále používam. Unit je vytvorený na základe určitých skúseností z iných unitov, je upravený pre naše účely a je freeware, môžete ho teda ľubovoľne používať alebo upravovať. Mnohí ho isto vylepšíte podľa seba.

Dúfam, že dnešné stretnutie s Pascalom vám bude užitočné aj vo vašej programátorskej praxi, za domácu úlohu preštudujte zdrojový text unitu a na budúce prejdeme na OOP.

Listing č. 1:

```
unit dbase3;
{$B-,V-,I-}

INTERFACE {-----------------------------------------------}

uses  dos;

const  MaxFelder  = 50;                { max poloziek }

type  FeldStr    = string[10];
        DbFeld     = record
        name : FeldStr;    { Meno polozky (VELKE PISMENA! }
        typ  : char;   { Typ polozky: C=String, N=Numericky }
        { Dlzka polozky}  size : byte;  { D=Datum, L=Boolean }
        nk   : byte;                   { desatinne miesta }
                  off        : word;
                  end;
    DbStruktur = record
                  felder     : word;      { pocet poloziek }
                  feld       : array[1..MaxFelder] of DbFeld;
                               { Zoznam poloziek }
                  datei      : file;      { I/O-Data }
                  j,m,t      : byte;     { Datum poslednej zmeny }
                  recs,                  { pocet zaznamov }
                  FPos       : longint;{ Ukazatel na data }
                  hdsize,                { velkost hlavicky+ 1}
                  rsize      : word;     { velkost dat+ 1 }
                  buff       : pointer;{ I/O-buffer }
                  modi       : boolean;{ znak zmeny }
                  end;

    DbPointer    = ^DbStruktur;
    PathStr      = string[79];

                                   { Result: > 0 = I/O-chyba }
var   DbResult    : integer;           { 0 = o.k. }
      DbEOF       : boolean;           { -1 = nespravne cislo vety}
                                  { -2 = nespravne meno polozky }

procedure DbUse(f:DbPointer; name:string);  {otvorenie databaze}
procedure DbClose(f:DbPointer);      { zatvorenie databazen }
procedure DbGo(f:DbPointer; p:longint);  { presun ukazatela }
procedure DbTop(f:DbPointer);        { skok na zaciatok baze }
procedure DbBottom(f:DbPointer);   { skok na koniec baze }
procedure DbSkip(f:DbPointer);       { 1 veta vpred }
procedure DbSkipBack(f:DbPointer);{ 1 vetu spat }

function DbRead(f:DbPointer; fname:FeldStr):string; {vycitanie
polozky}
procedure DbReadT(f:DbPointer; fname:FeldStr; var x); { dto.,
TP-Format }

procedure DbCreate(f:DbPointer; name:string);  { vytvorenie
databaze }
procedure DbAppend(f:DbPointer);  { naplnenie vety }
procedure DbReplace(f:DbPointer; fname:FeldStr; s:string);
{ prepis vety }
procedure DbReplT(f:DbPointer; fname:FeldStr; var x);  { dto.,
TP-Format }

procedure DbDelete(f:DbPointer);     { zmazanie vety }
procedure DbRecover(f:DbPointer);  { obnovenie vety }
function  DbDeleted(f:DbPointer):boolean;  { veta zmazana }

IMPLEMENTATION  {-----------------------------------------}

type   xa     = array[0..$ff00] of byte;

var       Header             : record
          ID,j,m,t           : byte;
          recs               : longint;
          hdsize,rsize       : word;
          dummy              : array[1..20] of byte;
          end;

 XFeld    : record
          name               : array[1..10] of char;
          dummy1             : byte;
          typ                : char;
          dummy2             : array[1..4] of byte;
          size,nk            : byte;
          dummy3             : array[1..14] of byte;
          end;

          FeldNr : word;

procedure FeldNummer(f:DbPointer; fname:string);
var i : word;
begin
  FeldNr:=1;
  while (FeldNr<=f^.felder) and
(f^.feld[FeldNr].name<>fname) do inc(FeldNr);
  if FeldNr>f^.felder then
    DbResult:=-2
  else
    DbResult:=0;
end;

{- Otvorenie databaze;  f   : ukazatel }
{-  name: meno databaze  }

procedure DbUse(f:DbPointer; name:string);

var       i  : word;
          l  : byte;
          o  : word;

begin
  fillchar(f^,sizeof(f^),0);
  if pos('.',name)=0 then name:=name+'.dbf';
  assign(f^.datei,name);
  reset(f^.datei,1);
  DbResult:=IOResult;
  if DbResult=0 then begin
    with f^ do begin
      blockread(datei,Header,SizeOf(Header));
      j:=header.j; m:=header.m; t:=header.t;
      recs:=header.recs; rsize:=header.rsize;
      felder:=(header.hdsize-$21) div $20;
      hdsize:=header.hdsize+1;
      o:=0;
```

```
      for i:=1 to felder do begin
        blockread(datei,XFeld,SizeOf(XFeld));
        l:=1;
        while (l<10) and (XFeld.name[l]<>#0) do inc(l);
        move(XFeld.name[1],feld[i].name[1],l-1);
        feld[i].name[0]:=chr(l-1);
        feld[i].typ:=XFeld.typ;
        feld[i].size:=XFeld.size;
        feld[i].nk:=Xfeld.nk;
        feld[i].off:=o;
        inc(o,XFeld.size);
        end;
      modi:=false;
      DbEOF:=(recs=0);
      getmem(buff,rsize);
      if not DbEOF then DbGo(f,1);
      FPos:=1;
      end;
    DbResult:=IOResult;
    end;
end;

{- Zatvorenie databaze;  f : ukazatel }

procedure DbClose(f:DbPointer);

var ja,mo,ta,wt : word;

begin
  with f^ do begin
    if modi then begin
      seek(datei,0);
      blockread(datei,Header,SizeOf(Header));
      GetDate(ja,mo,ta,wt);
      Header.j:=ja-1900; Header.m:=mo; Header.t:=ta;
      Header.recs:=recs;
      seek(datei,0);
      blockwrite(datei,header,SizeOf(Header));
      end;
    close(datei);
    freemem(buff,rsize);
    end;
  DbResult:=IOResult;
end;

{- Chod na vetu;  f : ukazatel}
{-               p : nova pozicia}

procedure DbGo(f:DbPointer; p:longint);
begin
  with f^ do
    if (p>recs) or (p<1) then
      DbResult:=-1
    else begin
      seek(datei,hdsize+(p-1)*rsize);
      blockread(datei,buff^,rsize);
      DbResult:=IOResult;
      FPos:=p;
      end;
end;

{- skok na zaciatok baze}

procedure DbTop(f:DbPointer);
begin
  with f^ do
      begin
      seek(datei,hdsize+(1-1)*rsize);
      blockread(datei,buff^,rsize);
      DbResult:=IOResult;
      FPos:=1;
      end;
end;

{- skok na koniec baze}

procedure DbBottom(f:DbPointer);
begin
  with f^ do
      begin
      seek(datei,hdsize+(recs-1)*rsize);
      blockread(datei,buff^,rsize);
      DbResult:=IOResult;
      FPos:=recs;
      end;
end;

{- posun o jednu vetu vpred}

procedure DbSkip(f:DbPointer);
begin
  with f^ do
    if FPos>=recs then
      DbEOF:=true
    else begin
      inc(FPos);
      seek(datei,hdsize+(FPos-1)*rsize);
      blockread(datei,buff^,rsize);
      end;
  DbResult:=IOResult;
end;

{- posun o jednu vetu spat }

procedure DbSkipBack(f:DbPointer);
begin
  with f^ do
    if FPos<1 then FPos:=recs
    else begin
      dec(FPos);
      seek(datei,hdsize+(FPos-1)*rsize);
      blockread(datei,buff^,rsize);
      end;
  DbResult:=IOResult;
end;

{- Vycitanie dat z vety }

function DbRead(f:DbPointer; fname:FeldStr):string;

var  h  : string;

begin
  with f^ do
    if FPos>recs then
      DbRead:=''
    else begin
      FeldNummer(f,fname);
      if DbResult<0 then DbRead:=''
      else begin
        move(xa(buff^)[feld[FeldNr].off],h[1],feld[FeldNr].size);
        h[0]:=chr(feld[FeldNr].size);
        DbRead:=h;
        DbResult:=0;
        end;
      end;
end;

{- vycitanie dat v Turbo-Pascal-Formate }
{-   f      : ukazatel }
{-   fname : meno polozky }
{-   x      : pascalovska premenna }

procedure DbReadT(f:DbPointer; fname:FeldStr; var x);
var h     : string;
    r     : real;
    res   : integer;
begin
  h:=DbRead(f,fname);
  with f^ do
    if h<>'' then
      case feld[FeldNr].typ of
        'C' : begin
                while (h<>'') and (h[length(h)]=' ') do
delete(h,length(h),1);
                string(x):=h;
              end;
        'N' : begin val(h,r,res); real(x):=r; end;
        'D' :
string(x):=copy(h,7,2)+'.'+copy(h,5,2)+'.'+copy(h,1,4);
        'L' : boolean(x):=(h='J') or (h='T');
      end;
end;

{- vytvorenie databaze;  f: ukazatel }
{-                fname: meno polozky }

procedure DbCreate(f:DbPointer; name:string);

var i : word;
```

```
    o : word;
    x : array[1..2] of char;

    ta,mo,ja,wt : word;

begin
  if pos('.',name)=0 then name:=name+'.dbf';
  assign(f^.datei,name);
  rewrite(f^.datei,1);
  DbResult:=IOResult;
  if DbResult=0 then begin
    with f^ do begin
      fillchar(Header,sizeof(Header),0);
      Header.ID:=3;
      getdate(ja,mo,ta,wt);
      Header.j:=ja-1900; Header.m:=mo; Header.t:=ta;
      Header.hdsize:=felder*$20+$21;
      rsize:=1;
      for i:=1 to felder do begin
        if feld[i].typ='D' then feld[i].size:=8;
        if feld[i].typ='L' then feld[i].size:=1;
        inc(rsize,feld[i].size);
        end;
      Header.rsize:=rsize;
      blockwrite(datei,Header,SizeOf(Header));
      hdsize:=header.hdsize+1;
      o:=0;
      for i:=1 to felder do begin
        fillchar(XFeld,sizeof(XFeld),0);
        move(feld[i].name[1],XFeld.name,length(feld[i].name));
        XFeld.typ:=feld[i].typ;
        XFeld.size:=feld[i].size;
        if feld[i].typ='N' then XFeld.nk:=feld[i].nk;
        blockwrite(datei,XFeld,SizeOf(XFeld));
        feld[i].off:=o;
        inc(o,feld[i].size);
        end;
      modi:=false;
      DbEOF:=true;
      getmem(buff,rsize);
      recs:=0; FPos:=1;
      x[1]:=^M; x[2]:=^Z;
      blockwrite(datei,x,2);
      end;
    DbResult:=IOResult;
    end;
end;

{- naplnenie vety }

procedure DbAppend(f:DbPointer);
begin
  with f^ do begin
    inc(recs);
    fillchar(buff^,rsize,' ');
    xa(buff^)[rsize]:=$1a;
    seek(datei,hdsize+(recs-1)*rsize-1);
    blockwrite(datei,buff^,rsize+1);
    FPos:=recs;
    modi:=true;
    end;
  DbResult:=IOResult;
end;

{- prepis polozky }

procedure DbReplace(f:DbPointer; fname:FeldStr; s:string);
begin
  with f^ do
    if (FPos>recs) or (FPos<1) then
      DbResult:=-3
    else begin
      FeldNummer(f,fname);
      if DbResult=0 then begin
        fillchar(buff^,rsize,' ');
        move(s[1],buff^,length(copy(s,1,feld[FeldNr].size)));
        seek(datei,hdsize+(FPos-1)*rsize+feld[FeldNr].off);
        blockwrite(datei,buff^,feld[FeldNr].size);
        modi:=true;
        DbResult:=IOResult;
        end;
      end;
end;

{- prepis polozky v Turbo-Pascal-Formate }

procedure DbReplT(f:DbPointer; fname:FeldStr; var x);   { dto.,
TP-Format }
var h : string;
begin
  FeldNummer(f,fname);
  if DbResult=0 then
    with f^ do begin
      case feld[FeldNr].typ of
        'C' : h:=string(x);
        'N' : str(real(x):feld[FeldNr].size:feld[FeldNr].nk,h);
        'D' : begin
                h:=string(x);
h:=copy(h,7,4)+copy(h,4,2)+copy(h,1,2);
              end;
        'L' : if boolean(x) then h:='T' else h:='F';
      end;
      DbReplace(f,fname,h);
      end;
end;

procedure SetDelFlag(f:DbPointer; c:char);
begin
  with f^ do begin
    if (FPos<1) or (FPos>recs) then exit;
    seek(datei,hdsize+(FPos-1)*rsize-1);
    blockwrite(datei,c,1);
    DbResult:=IOResult;
    end;
end;

{- vymaz vety }

procedure DbDelete(f:DbPointer);
begin
  SetDelFlag(f,'*');
end;

{- obnova vety }

procedure DbRecover(f:DbPointer);
begin
  SetDelFlag(f,' ');
end;

{- je veta zmazana?}

function DbDeleted(f:DbPointer):boolean;
var c : char;
begin
  with f^ do
    if (FPos<1) or (FPos>recs) then DbDeleted:=false
    else begin
      seek(datei,hdsize+(FPos-1)*rsize-1);
      blockread(datei,c,1);
      DbResult:=IOResult;
      DbDeleted:=(c='*');
      end;
end;

END.
```

Listing č. 2:

```
program DBDemo; {demonstruje vytvorenie a naplnenie DBF
suboru}

uses crt,dbase3;

var f      : DbPointer;
    dnes   : string[10];
    numer  : real;
    hotovo : boolean;

BEGIN
  clrscr;
  new(f);
  with f^ do
   begin
    felder:=3;
    feld[1].name:='RETAZEC';  feld[1].typ:='C';feld[1].size:=10;
    feld[2].name:='CISLO'; feld[2].typ:='N'; feld[2].size:=5;
feld[2].nk:=2;
    feld[3].name:='LOGIKA'; feld[3].typ:='L';
    DbCreate(f,'TEST'); {vytvorenie suboru s menom TEST.DBF}

    DbAppend(f);        {vytvorenie 1.vety a jej naplnenie}
    dnes:='PCrevue'; numer:=1000; hotovo:=false;
```

```
    DbReplT(f,'RETAZEC',dnes);
    DbReplT(f,'CISLO',numer);
    DbReplT(f,'LOGIKA',hotovo);

    DbAppend(f);        {vytvorenie 2.vety a jej naplnenie}
    dnes:='TPascal'; numer:=22; hotovo:=true;
    DbReplT(f,'RETAZEC',dnes);
    DbReplT(f,'CISLO',numer);
    DbReplT(f,'LOGIKA',hotovo);

    DbAppend(f);        {vytvorenie 3.vety a jej naplnenie}
    dnes:='01.10.1996'; numer:=3.75; hotovo:=false;
    DbReplT(f,'RETAZEC',dnes);
    DbReplT(f,'CISLO',numer);
    DbReplT(f,'LOGIKA',hotovo);

    DbClose(f);          { uzavretie databaze}
  end;
END.
```

Listing č. 3:

```
program DBList; {vypise polozky a obsah poloziek DBF suboru}

uses crt, dbase3;

var name : string;
    i    : integer;
    f    : DbPointer;

BEGIN
  clrscr;
  if paramcount=1 then name:=paramstr(1)
   else begin
          write('Meno databaze: ');
          readln(name); writeln;
        end;
  new(f);
  DbUse(f,name);   {otvorenie databaze}
  if DbResult<>0 then writeln ('Nezname meno suboru!')
    else with f^ do
      begin
        writeln('Struktura vety');
        writeln('---------------');
        for i:=1 to felder do
          begin
            write(i:2,'. ',copy(feld[i].name+'          ',
                        1,11),feld[i].typ,' ',feld[i].size);
            if feld[i].typ='N' then writeln(':',feld[i].nk)
             else writeln;
          end;
        writeln;
        writeln('Obsah datovych viet:');
        writeln('-------------------');
        while not DbEOF do
          begin
            write(FPos:4,'. ');
            for i:=1 to felder do write(DbRead(f,feld[i].name),' ');
            writeln;
            DbSkip(f);
          end;
        writeln;

        write('Skok na prvy zaznam     : ');
        DbTop(f);
        for i:=1 to felder do write(DbRead(f,feld[i].name),' ');
        writeln;

        write('Skok na posledny zaznam : ');
        DbBottom(f);
        for i:=1 to felder do write(DbRead(f,feld[i].name),' ');
        writeln;

        write('Posun o vetu spat       : ');
        DbSkipBack(f);
        for i:=1 to felder do write(DbRead(f,feld[i].name),' ');
        writeln;

        write('Pocet viet              : ');
        writeln(recs);

        DBClose(f); {Zatvorenie databaze}
      end;
END.
```

Miroslav Oravec