OZOGAN KLONDAIK 2

Omezení demoverze:
- činnost systému je po 10 minutách ukončena. Potom je nutné program KLONDAIK znovu spustit.

- délka editovaného programu je maximálně 50 řádků
- nelze vytisknout graf a tabulku
- nelze vytvořit novou databázi, editovat strukturu databází
- u databázové sestavy je možné tisknout pouze první stranu

Poslední informace a aktualizace ze dne 16/04/1998.

10. NOVINKY A ZMĚNY V PROGRAMU

Program se stále vyvíjí a doplňuje o nové funkce. Veškeré změny jsou zde zahrnuty v přehledu, podrobněji potom v kompletní nápovědě.

VERZE 2.02 (16/04/98)

Nové systémové procedury umožní definovat vlastní menu aplikace. V uživatelském menu představuje přitom volba jedna položka spuštění dosavadního programu. Podrobněji viz 63. lekce kurzu obsluhy a použití systému, případně popis nových procedur:

MenuInit - inicializuje (nuluje) menu
MenuChange - nastaví položku uživatelského menu
MenuIcon - nastaví obsah ikony uživatelského toolbaru
MenuActive - aktivuje/deaktivuje uživatelské menu

Systém byl doplněn o možnost zakódovat programy, které je potom možné například spouštět i pod demoverzí systému bez obvyklých omezení demoverze. Viz 64. lekce kurzu obsluhy a použití systému.

Pokud se načte do systému program, k němuž existuje soubor shodného jména ale s příponou *.TXT, je tento soubor načten do textového okna a zobrazen současně se zdrojovým kódem programu. Je proto možné takto automaticky zobrazit ke každému programu ihned i příslušnou dokumentaci (pokud existuje).

VERZE 2.01 (09/03/98)

Kromě opravy jedné chyby v proceduře DbfAppendFrom byly doplněny nové lekce 47 až 62 kurzu obsluhy a použití systému, byla doplněna možnost editace memo položek databází (viz 47. lekce kurzu obsluhy programu) a několik nových procedur a funkcí:

SYSTÉMOVÁ KNIHOVNA:

FormAllHide - minimalizuje všechna otevřená okna
CommandForm - změna velikosti a umístění příkazového okna
ConsoleForm - změna velikosti a umístění výstupního textového okna
DbfForm - změna velikosti a umístění databázového okna
ChartForm - změna velikosti a umístění okna s grafem
ImageForm - změna velikosti a umístění grafického výstupního okna
ProgramForm - změna velikosti a umístění okna s programem
SpreadForm - změna velikosti a umístění okna s tabulkou

KNIHOVNA PRO PRÁCI S DATABÁZEMI:

DbfCopyToInfo - kopíruje strukturu databáze do informační databáze
DbfCreateFrom - založí novou databázi z informační databáze
DbfCreateInfo - založí novou prázdnou informační databázi
DbfFieldCount - vrací počet položek (sloupců) databáze
DbfFieldDec - vrací počet desetinných míst zadané položky
DbfFieldName - vrací jméno položky dle pořadového čísla
DbfFieldType - vrací typ databázové položky (N/C/L/M)
DbfFieldWidth - vrací délku databázové položky

KNIHOVNA PRO PRÁCI S ŘETĚZCI:

Day	- vrací jméno dne v týdnu
LowerCase	- konvertuje řetězec na malá písmena
Month	- vrací jméno měsíce v roce
Space	- vrací prázdný řetězec zadané délky
Trim	- uřízne z řetězce počáteční a koncové mezery
TrimLeft	- uřízne z řetězce počáteční mezery
TrimRight	- uřízne z řetězce koncové mezery
UpperCase	- konvertuje řetězec na velká písmena

VERZE 2.00 (21/01/98)

V programu provedeny velmi rozsáhlé změny - doplnění možnosti zpracování databází formáru FoxPro, tabulek formátu Excel 4.00 a grafů.

Sestavené a odladěné programy je možné spouštět i mimo vývojové prostředí systému KLONDAIK jako výkonná makro windows. To lze provést tak, že definujete ve Windows asocici souborů s příponou *.IPS na systém KLONDAIK. Pokud potom kliknete v manažeru souborů na program vytvořený v systému KLONDAIK, bude spuštěna ihned interpretace programu bez zobrazení vývojového prostředí. Pokud takovýto program ukončíte příkazem QUIT, bude ukončen nejen spuštěný program, ale současně i vývojové prostředí. Takto je možné naprogramovat inteligentní výkonná makra systému Windows (obdoba *.BAT souborů v systému MS-DOS). Výkonná makra lze ve spojení s možností spouštění externích programů výhodně použít na příklad i pro automatizaci

Možnost spouštění programu jako výkonné makro Windows je v demoverzi programu zablokována.

Připravuje se (zatím nelze) možnost napsaný program zakódovat do nečitelného tvaru a ten potom spouštět mimo vývojové prostředí KLONDAIK ve stulu výše uvedených výkonných maker Windovs. Výhodou bude nejen ochrana programu před neoprávněnými zásahy do jeho činnosti, ale hlavně možnost provozovat takovýto program přímo nad demoverzí systému KLONDAIK. Tím se stane demoverze systému KLONDAIK runtime modulem vytvořených programů bez nutnosti jejich spouštění pod plnou verzí programu. Zakódované programy spuštěné v demoverzi systému nebudou samozřejmě omezovány funkčností demoverze (délka programu, časové omezení apod.). Uvedená možnost kódování programu bude do programu zabudována na začátku roku 1998 a registrovaní uživatelé získají uvedenou úpravu zdarma.

SYSTÉMOVÁ KNIHOVNA:

PortInput	- čte hodnotu z portu
PortOutput	- zapíše hodnotu na port
RunPrg	- spustí externí program *.EXE
Quit	- ukončí činnost systému

INTERAKTIVNÍ KNIHOVNA:

Doplněny jsou procedury a funkce pro zvýšení účinnosti formulářů s možností programování her na ploše formuláře.

AddFrame	- definuje na formuláři rámeček
AddIcon	- definuje na formuláři zobrazení ikony (bitmapy)
AddSpeedButton	- definuje na formuláři tlačítko s bitmapou
FormActionKey	- definuje proceduru pro zpracování stisknuté klávesy
ChangeIcon	- změní zobrazovanou ikonu (bitmapu)
MoveIcon	- změní polohu ikony (bitmapy) na formuláři
MoveButton	- změní polohu tlačítka na formuláři

KNIHOVNA PRO ZOBRAZENÍ GRAFŮ:

V knihovně pro zobrazování grafů jsou definovány procedury a funkce sloužící k zobrazování grafů, zadávání hodnot a nastavení zobrazení grafu.

Chart3 - přepínání zobrazení 2D a 3D grafu
ChartData - definuje data grafu
ChartGrid - definuje použití vodorovné a svislé mřížky
ChartHide - minimalizuje okno s grafem do ikony
ChartInit - inicializuje graf, definuje počet hodnot
ChartLegend - definuje legendu ke grafu (popis os)
ChartLoad - načte graf ze souboru z disku
ChartSave - uloží graf do souboru na disk
ChartShow - obnoví původní velikost okna s grafem
ChartTitle - definuje nadpisy grafu
ChartType - změna typu grafu
ChartUpdate - obnoví zobrazení grafu po aktualizaci hodnot

KNIHOVNA PRO PRÁCI S TABULKAMI:

V knihovně pro práci s tabulkami jsou definovány procedury a funkce sloužící k práci s tabulkami ve formátu Excel verze 4, případně ve vlastním, optimalizovaném formátu.

SpreadAutoRecalc - aktivuje/deaktivuje okamžitý přepočet tabulky
SpreadClear - vymaže obsah tabulky
SpreadCol - aktivuje zadaný sloupec
SpreadFormula - uloží do aktivní buňky zadaný vzorec
SpreadGetMaxCol - vrací počet sloupců tabulky
SpreadGetMaxRow - vrací počet řádků tabulky
SpreadGetNumber - přečte číselnou hodnotu z aktivní buňky
SpreadGetText - přečte textovou hodnotu z aktivní buňky
SpreadHide - minimalizuje okno s tabulkou do ikony
SpreadInit - inicializuje novou tabulku
SpreadLoad - načte zadaný soubor jako tabulku
SpreadNumber - uloží do aktivní buňky zadané číslo
SpreadRecalc - přepočítá tabulku podle zadaných vzorců
SpreadRow - aktivuje zadaný řádek
SpreadSave - uloží tabulku do souboru na disk
SpreadShow - obnoví původní velikost okna s tabulkou
SpreadShowColHead - aktiavace/deaktivace zobrazení záhlaví sloupců
SpreadShowLines - aktivace/deaktivace zobrazení mřížky tabulky
SpreadShowRowHead - aktiavace/deaktivace zobrazení záhlaví řádků
SpreadText - uloží do aktivní buňky zadaný text

KNIHOVNA PRO PRÁCI S DATABÁZEMI:

Knihovna je určena pro práci s databázemi ve formátu FoxPro.

DbfAppendBlank - přidá prázdný záznam do databáze
DbfAppendFrom - načte (přihraje) data z jiné databáze
DbfBof - test začátku databázového souboru
DbfCopyFile - kopíruje databázi do nového souboru
DbfCopyToText - kopíruje databázi do textového souboru
DbfCopyStru - kopíruje strukturu databáze do nového souboru
DbfContinue - pokračuje v hledání dalšího záznamu(viz DbfLocate)
DbfCount - vrací počet záznamů dle nastaveného filtru

DbfDelete - označí záznam ke zrušení
DbfDeleted - vrací informaci, zda je záznam určen ke zrušení
DbfEvalLog - vrací výsledek logického databázového výrazu
DbfEvalNum - vrací výsledek numerického databázového výrazu
DbfEvalStr - vrací výsledek řetězcového databázového výrazu
DbfEvalTest - testuje správnost databázového výrazu
DbfEof - test konce databázového souboru
DbfFound - vrací informaci, zda bylo hledání úspěšné
DbfGo - přesun na zadané číslo záznamu
DbfGoBottom - přesun na konec databázového souboru
DbfGoTop - přesun na začátek databázového souboru
DbfIndexTag - založí nový index databáze
DbfLocate - hledá záznam dle zadané podmínky
DbfPack - fyzicky zruší záznamy označené k výmazu
DbfRecall - obnoví platnost záznamu určeného ke zrušení
DbfRecCount - vrací celkový počet záznamů v databázi
DbfRecNo - vrací číslo aktuálního záznamu
DbfReindex - obnoví indexování databáze
DbfSeek - hledá v databázi dle zadaného klíče
DbfName - vrací název souboru databáze
DbfReadDat - přečte datovou položku z databáze
DbfReadLog - přečte logickou položku z databáze
DbfReadNum - přečte mumerickou položku z databáze
DbfReadStr - přečte řetězcovou položku z databáze
DbfReport - vytiskne sestavu
DbfSelect - aktivace pracovní databázové oblasti
DbfSetDeleted - nastaví, přístupnost zrušených záznamů
DbfSetFilter - nastaví filtr databáze (výběr dle hodnot)
DbfSetOrder - určí, který index bude aktivní
DbfTagArea - dle zadaného jména indexu vrací jeho číslo
DbfTagName - dle zadaného čísla indexu vrací jeho název
DbfSkip - přesun ukazatele v databázovém souboru
DbfUse - otevře databázi
DbfWriteDat - zapíše datovou hodnotu do položky databáze
DbfWriteLog - zapíše logickou hodnotu do položky databáze
DbfWriteNum - zapíše numerickou hodnotu do položky databáze
DbfWriteStr - zapíše řetězcovou hodnotu do položky databáze
DbfZap - fyzicky zruší všechny záznamy databáze

1.0. Úvod

Systém OZOGAN KLONDAIK je integrovaný vývojový systém pro zápis a provoz programů ve vlastním jazyce INTER-PASCAL, velmi podobném jazyku PASCAL. Programy vyvíjené v systému nejsou kompilovány, ale interpretovány ve stylu jazyka BASIC. Skloubením interpretu s jazykem typu PASCAL vznikl výkonný prostředek dovolující snadný vývoj a ladění programů. Systém je proto určen všem, kteří chtějí začít programovat ve Windows a mají zájem se naučit základům jazyka PASCAL. Výhodně poslouží i pro výuku programování. Lze v něm však tvořit i jednoduché aplikace a nahrazuje jazyk BASIC dodávaný donedávna s operačním systémem MS-DOS.

Systém OZOGAN KLONDAIK je vybaven nástroji pro ladění programů. V ladícím režimu je možné editovaný program krokovat po jednotlivých řádcích programu s případným přeskokem již odladěných procedur. Současně je možné nechat si zobrazit obsah zadaných proměnných, případně změnit jejich hodnoty. Program lze spustit v režimu animace, kdy se v okně s programem aktuálně zobrazují vykonávané řádky programu.

1.1. Základní pojmy

Protože bude v dalším textu uváděno několik ne zcela běžné názvosloví, uvádíme vysvětlení několika důležitých pojmů a vysvětlivek.

Systém představuje vývojové prostředí OZOGAN KLONDAIK, ve kterém jsou editovány a provozovány uživatelské programy. Systém OZOGAN KLONDAIK je interpret, stejně jako jazyk BASIC.

Program označuje uživatelův program napsaný ve vnitřním jazyce INTER-PASCAL. Program je možné v systému editovat, ladit a spouštět. Programy odladěné v systému OZOGAN KLONDAIK nejsou zpravidla bez dalších úprav spustitelné v běžném jazyce Pascal.

Interpret je překladač, který netransformuje zdrojové programy na ekvivalentní programy ve strojovém kódu, ale sám zajišťuje provedení programu jeho interpretací. Interpretační překladač tedy přímo interpretuje zdrojový program, tj. čte instrukce, okamžitě je překládá a poté ihned provede určené akce.

Interpretace znamená provozování programu tak, že každá řádka zdrojového programu je systémem samostatně přeložena a ihned provedena.

Jazyk INTER-PASCAL je podmnožinou standardního jazyka Pascal. Obsahuje však některé změny a úpravy, díky kterým je v mnoha případech jednodušší a snažší. Pokud znáte základy jazyka Pascal, nebude pro vás problém naučit se používat jazyk INTER-PASCAL. Naopak, po zvládnutí jazyka INTER-PASCAL bude pro Vás snadné přejít na standardní zápis programů v jazyce Pascal. Jazyk vychází ve své syntaxi z klasického jazyka Pascal. Struktura zápisu programu však byla způsobem zápisu struktury programu ve stylu databázových jazyků typu Dbase a FoxPro zjednodušena a zpřehledněna. Některé drobné prvky jazyka byly převzaty z jazyků Basic a C. Výsledkem by měl být nový jazyk (verze jazyka?), který může být sice mnohými odborníky a recenzenty odsuzován, běžný uživatel, pro kterého je program určen jej však jistě uvítá.

Systémové požadavky - program je určen pro operační systém Windows 3.1 a výše (Windows 95), je provozovatelný na počítačích od PC 386, 8 Mb paměti RAM. Výhodnější je použití počítačů s mikroprocesorem Pentium, 16 Mb paměti RAM.

2.0 Okna systému OZOGAN KLONDAIK

Systém OZOGAN KLONDAIK obsahuje ve svém integrovaném prostředí všechny základní prostředky pro zápis a editaci programu, interpretaci kódu a krokování programu. Všechny informace jsou proto rozděleny do několika oken, ve kterých veškerá uvedená činnost probíhá.

2.1. Okno příkazy

Okno zpřístupňuje velmi výhodnou vlastnost, to je možnost zadávání příkazů, které jsou ihned provedeny. Běžný jazyk Pascal toto neumožňuje, protože je kompilátor. Systém OZOGAN KLONDAIK je však interpret a tak není pro něj problém provést jednotlivé příkazy bez nutnosti kompilace programu. Díky tomu bude pro vás jednodušší proniknout do tajů jazyka Pascal, protože budete mít možnost si většinu příkazů, procedur a funkcí vyzkoušet bez nutnosti psaní programu. Další využití je možnost práce s deklarovanými proměnnými v ladícím režimu.

Příkazové okno představuje seznam příkazů, které zadáváte systému z klávesnice. Vzhledově připomíná editor. Pokud ale po zápisu příkazu stisknete klávesu ENTER, pokusí se systém řádku, na které stojí kurzor interpretovat. Kurzor potom přejde na novou, prázdnou řádku. Pokud dokonce stisknete klávesu ENTER v okamžiku, kdy máte kurzor uprostřed slova, nedojde k ukončení řádku, jak bývá zvykem u běžných editorů. Systém převezme opět celou řádku, přeloží a provede zadané příkazy.

Pokud budete chtít zadat příkaz, který je již uveden v seznamu příkazů, stačí umístit kurzor na požadovanou řádku a stisknout klávesu ENTER. Příkaz se provede a pokud byl předchozí příkaz jiný, zkopíruje se řádek na konec seznamu. Pokud by jste chtěli v některém řádku pouze provést změnu, stačí najet kurzorem na řádku, provést úpravu a stisknout ENTER. Původní řádek zůstane zachován beze změny a nový řádek se zařadí opět na konec seznamu.

V příkazovém okně je možné zadávat většinu příkazů, procedur a funkcí jazyka INTER-PASCAL. Nelze používat víceřádkové příkazy a testování podmínek (FOR..ENDFOR, REPEAT..UNTIL, IF..ENDIF, apod.). Není také možné z příkazového okna deklarovat proměnné. Pokud ale spustíte program, ve které jsou proměnné deklarovány, zůstane zachována jejich deklarace i po ukončení programu. Stejně tak je možné definovat uživatelské procedury a funkce, které jsou opět přístupné až po spuštění programu.

Vlastnosti příkazového okna využijete především v ladícím režimu, kdy máte možnost změny hodnot proměnných, výpisu jejich obsahu, případně i zadávání nutných příkazů, které nejsou v programu uvedeny. To vám umožní velmi efektivní styl práce při vývoji programu. Pokud později přejdete na běžný jazyk Pascal, bude vám příkazové okno určitě chybět.

2.2. Okno program

Okno obsahuje program, se kterým systém aktuálně pracuje. Jedná se vlastně o editor zdrojového textu programu. Z programu je možné okno pouze ukrýt příkazem ProgramHide a zpětně obnovit příkazem ProgramShow.

Po volbě menu 'nový program' je do okna umístěna základní minimalizovaná kostra programu. Při zápisu programu jsou dostupné běžné základní editační funkce.

Potřebujete-li převést část jiného programu do editovaného programu, můžete tak učinit pomocí okna TEXT, ve kterém si otevřete program, ze kterého chcete část zdrojového textu převzít. Požadovanou část programu potom označíte a převedete do schránky. Po přepnutí na okno PROGRAM potom přesunete obsah schránky na požadované místo.

Program může mít délku maximálně 32 kB a je na disk ukládán v souboru s příponou *.IPS. Celý program včetně uživatelsky definovaných procedur musí být uložen v jednom souboru. Program není nijak kódován, je čitelný běžnými textovými editory. Text programu je možné vytisknou volbou z menu, případně pomocí ikony z toolbaru.

Po spuštění programu je v režimu animace programu v okně s programem zobrazována aktuální vykonávaná řádka programu. V ladícím režimu se v okně s programem také zobrazuje aktuální vykonávaná řádka programu, je zde však možné mezi jednotlivými kroky programu listovat zdrojovým textem.

2.3. Okno výstup-grafika

Okno představuje jednoduchý grafický editor a slouží pro výstup grafických funkcí jazyka INTER-PASCAL. Kromě možnosti ukrytí okna procedurou ImageHide a jeho zobrazení procedurou ImageShow dovoluje také definovat velikost plochy procedurou ImageInit a vymazat obsah plochy procedurou ImageClear. Mnoho dalších zabudovaných procedur slouží pro nastavení fontu, barvy, stylu a síly čáry a kreslení do okna. Možné je také uložit obsah grafického okna do souboru typu *.BMP procedurou ImageSave, a nahrát procedurou ImageLoad soubor typu *.BMP do grafického okna, případně vytisknout obsah grafického okna na tiskárnu.

Grafické okno vám umožní naprogramovat požadovaný obrázek, případně programově měnit parametry zobrazovaných částí obrazu. Grafické okno je velmi výhodné pro první pokusy s programováním, protože umožňuje velmi jednoduchou a názornou formou prezentovat výsledky výpočtů.

Grafické okno lze mimo příkazů z programu ovládat i myší. Příslušné ikony se zobrazí vždy při aktivaci grafického okna. Pomocí ikon lze vymazat plochu, načíst obrázek ze souboru, uložit jej do souboru, případně vytisknout. Dalšími ikonami je možné definovat barvu a styl kreslených čar a ploch.

Podrobněji o možnostech přístupu do grafického výstupního okna z programu viz grafická knihovna.

2.4. Okno výstup-text

Okno slouží pro výstup textových informací ze spuštěného programu nebo příkazů z příkazového okna. Zápisy do okna se provádí příkazem WRITE (případně WRITELN). Obsah okna nelze nijak editovat, je možné jej pouze vytisknout, uložit obsah do textového souboru, případně zrušit jeho obsah.

Z programu je možné zabudovanými procedurami ovládat ukrytí okna a jeho opětné zobrazení procedurami ConsoleHide a ConsoleShow. Důležitější však je možnost zrušit obsah okna procedurou ConsoleClear, vytisknout obsah okna na tiskárnu procedurou ConsolePrint a uložit obsah výstupního okna do textového souboru procedurou ConsoleSave. Uvedené zabudované procedury jazyka INTER-PASCAL vám výhodně poslouží především pro možnost trvalého uchování vypočtených a zobrazených údajů. Současně tak eliminují nemožnost přímého tisku na tiskárnu příkazem WRITE.

Ikony pro práci s oknem se zobrazí při práci v editačním, případně ladícím režimu v horní liště okna po jeho aktivaci. Jedná se o tři ikony, které umožňují základní operace s oknem. První ikona provede po kontrolním dotazu výmaz obsahu okna. Druhá ikona vyvolá dialog pro zadání jména souboru, do kterého bude obsah okna uložen v textovém tvaru. Třetí ikona zajistí svým dialogem tisk obsahu okna na tiskárnu.

2.5. Okno databáze

Databázové okno slouží pro přímou práci s databázovým souborem typu FoxPro. Data se zobrazují přehledně v tabulce, kterou lze listovat všemi směry. Pokud obsahuje databáze indexy typu *.CDX, jsou otevřeny spolu s datovým souborem a je možné výběrovým boxem zadat třídící index databáze. V databázi lze vyhledávat dle klíče, nastavit filtr výběru záznamů dle zadané podmínky.

Pro práci s databázemi v okně slouží mimo několika ikon pro základní operace s databází i PopUp menu, které se zobrazí po stisku pravého tlačítka myši na ploše databázového okna. Do menu jsou zahrnuty náročnější operace s databázemi, jako je nastavení možnosti editace dat, nastavení filtru, hledání dle klíče, přidávání indexu, přepínání v datových oblastech, vytvoření a změna databáze.

Databázi je možné tisknout ve formě sestav. K návrhu sestav slouží vizuální generátor sestav s možností zadávání skupinování tisku, součtování položek a podobně.

Systém obsahuje mnoho procedur a funkcí pro přímou práci s databázemi. Některé procedury a funkce používají pro přímou práci s položkami databáze výrazy v syntaxi jazyků xBase (Dbase, Foxpro, Clipper).

V demoverzi programu není možné vytvářet nové databáze, ani měnit strukturu existujících databází.

Podrobněji o možnostech přístupu do databázového okna z programu viz procedury a funke pro práci s databázemi.

2.6. Okno tabulka

Zpracovávaná tabulka je v mnohém shodná s tabulkami, které obsahuje program Excel, verze 4. Pro podrobnější možnost seznámení se s možnostmi zpracování tabulek můžete proto použít příslušnou literaturu.

Tabulku představují sloupce a řádky. Sloupce jsou označovány písmeny od A (A, B, C, .., Z, AA, AB, ...). Řádky tabulky jsou číslovány. V tabulce může být maximálně 256 sloupců a 16384 řádků. Po spuštění systému je přitom nastavena velikost tabulky na 10 řádků x 10 sloupců. Rozdělením tabulky na slupce a řádky vzniknou v tabulce buňky. Buňky v tabulce se označují podle řádků a sloupců. Nejprve se uvádí sloupec, potom řádek. První buňka v prvním řádku se označuje A1, třetí buňka ve dvacátém řádku C20. Pokud se budete na buňky odkazovat přímo z programu, budete muset používat místo označování sloupců tabulky čísla. Například sloupec A musíte uvést jako 1, pro sloupec D použijete číslo 4 apod.

Do buňek tabulky je možné ukládat nejen čísla, ale i vzorce pro výpočty a texty. Pokud budete chtít, aby byl v některé buňce tabulky zobrazován výpočet, zadáte do buňky vzorec, podle kterého se má výsledek vypočítat. Ve vzorci se přitom můžete odkazovat na ostatní buňky tabulky. Vzorec pro výpočet hodnoty v buňce musí začínat znakem = (rovná se), za kterým musí následovat funkce pro výpočet a v závorkách uvedené parametry funkce. Budete-li chtít například chtít, aby byl v buňce A4 uveden výsledek výpočtu součtu buněk A1, A2 a A3, musíte do buňky A4 zadat vzorec =SUM(A1+A2+A3). Pokud by jste potřebovali součtovat větší počet buněk, bylo by značně pracné vypisovat samostatně každou buňku. Proto je možné zadat zjednodušeně rozsah buněk od počáteční po konečnou, oddělených dvojtečkou. Výše uvedený výpočet by se proto mohl napsat také následujícím způsobem =SUM(A1:A3). Pokud provedete přesun buňky obsahující vzorec na jiné místo, provede se automaticky úprava vzorce pro nové umístění.

Tabulku můžete mnoha způsoby graficky upravit a ztvárnit. Myší můžete upravovat velikost řádků a sloupců. To se provede tak, že po najetí kurzorem myši na čáru v záhlaví tabulky oddělující jednotlivé řádky a sloupce máte možnost po kliknutí levým tlačítkem myši a následném táhnutí v požadovaném směru upravit velikost řádků a sloupců.

S obsahem tabulky můžete rozsáhle manipulovat po vybrání části tabulky. To provede tak, že kliknete levým tlačítkem myši na plochu tabulky, podržíte stisknuté tlačítko myši a táhnutím označíte požadovanou oblast tabulky. Vybraná oblast je zvýrazněna barevně. S vybranou oblastí můžete potom dále pracovat. Uchopením za okraje můžete přesunout vybranou oblast na nové místo, přičemž se provede automaticky úprava vzorců na nové umístění. Další možností je použití PopUp menu, které se zobrazí po stisku pravého tlačítka myši nad tabulkou. Zobrazí se vám volby pro kopírování označené oblasti přes schránku Windows, případně její výmaz. Dalšími položkami PopUp menu můžete v tabulce ve vybrané oblast nastavit požadovaný font, okraje rámečků, zarovnávání obsahu buněk a vybarvení plochy buněk. Nabídky jsou zatím anglicky, v dalších verzích bude přepracováno do češtiny.

Tabulku si pro použití ve vašem programu můžete nejprve připravit do požadované podoby, uložit ji do souboru a potom ji z programu načíst a následně zadávat proměnné hodnoty tabulky. Tabulku lze však také vytvořit celou včetně definice vzorců pro výpočet přímo v programu.

Vzorce pro výpočet tabulek jsou obdobné s tabulkami typu Excel verze 4 (anglická varianta). Pro podrobnější seznámení se s používanými vzorci můžete proto použít příslušnou literaturu. Následuje pouze výběr některých možných funkcí pro výpočty ve vzorcích. V tabulce je možné používat například veliké množství trigonometrických, stastistickýh a finančních výpočtů. V konečné verzi programu budou funkce poprány podrobněji.

ABS - vrací absolutní hodnotu čísla

```
DATE      - vrací pořadové číslo zadaného dne
DAY       - převede pořadové číslo dne na den v měsíci
DATEVALUE - převede datum z řetězce na pořadové číslo
FACT      - vrátí faktoriál čísla
INT       - zaokrouhlí číslo na nejbližší menší celé číslo
LEN       - vrátí počet znaků textového řetězce
MID       - vrátí zadaný počet znaků z retězce od zadané pozice
MOD       - vrátí zbytek po dělení čísla
MONTH     - převede pořadové číslo dne na měsíc
RAND      - vrátí náhodné číslo z intervalu 0 až 1
REPLACE   - nahradí znaky v textu
ROUND     - zaokrouhluje číslo na zadaný počet číslic
SEARCH    - hledá v textu zadaný podřetězec
SUM       - sečte zadané argumenty
SUBSTITUE - nahradí v textu zadaný řetězec jiným
TIME      - vrací pořadové číslo zadaného času
TODAY     - vrátí pořadové číslo na den v týdnu
TRIM      - odstraní nadbytečné mezery v textu
TRUNC     - odřízne desetinnou část čísla
VALUE     - převádí textový argument na číslo
WEEKDAY   - převádí pořadové číslo na den v týdnu
YEAR      - převádí pořadové číslo na rok
```

Vzorce je možné výhodně použít i pro výpočty, které neobsahuje přímo jazyk INTER-PASCAL. To můžete provést tak, že inicializujete tabulku potřebné velikosti, do které doplníte vzorec pro výpočet, dosadíte vstupní hodnoty a po přepočtu tabulky načtete výsledek výpočtu.

Podrobněji o možnostech přístupu do okna s tabulkou z programu viz procedury a funke pro práci s tabulkami.

2.7. Okno grafy

Okno se používá pro grafické zobrazení změny sledovaných hodnot. Je možná změna nastavení grafu interaktivně přímo v okně pomocí ikon v podřízeném toolbaru. Zadání hodnot je ve vývojové betaverzi možné pouze z programu procedurou ChartData, případně stejnou procedurou z příkazového okna. V dalších verzích programu bude dopracována možnost změny hodnot grafu přímo v okně v editačním režimu grafu.

Podrobněji o možnostech přístupu do okna s grafem z programu viz procedury a funke pro práci s grafy.

2.8. Okno sledování proměnných

Okno se používá v ladícím režimu, případně v režimu animace programu. V okně je možné definovat seznam sledovaných proměnných, jejichž obsah se bude pravidelně po provedení každé řádky programu obnovovat. Po přepnutí do okna budou zobrazeny dvě ikony, které umožňují zápis a výmaz sledovaných proměnných.

Zápis proměnné do okna nemá žádný vliv na její obsah. Možné je i zapsat proměnnou, která není v programu deklarována. Zápisem proměnné do okna sledování se však neprovádí její deklarace. Pokud není proměnná deklarována, zobrazí se v okně pro sledování proměnných u hodnoty proměnné poznámka, že hodnota proměnné není definována. Nezaměňujte proto zápis proměnné do okna sledování s deklarací proměnné v programu !

Pokud použijete ikonu pro výmaz proměnné, dojde k jejímu zrušení pouze v okně sledování proměnných. Proměnná tímto nebude zrušena, ani nedojde ke změně její hodnoty.

Proměnné můžete do okna zadávat kdykoliv, stejně tak je můžete kdykoliv z okna sledování vymazat. Použití okna pro sledování proměnných má však význam pouze v ladícím režimu, případně v režimu animace programu. V jiných režimech není stav hodnot proměnných aktualizován.

Pokud se nacházíte v ladícím režimu, můžete provést z příkazového okna změnu hodnoty proměnné přiřazením její nové hodnoty. Do tabulky sledování hodnot proměnných není možné zadávat požadavek na zobrazení tzv. vypočtených položek (například a+b). Okno pro sledování proměnných nelze z provozovaného programu žádným způsobem ovládat.

2.9. Okno texty

Okno TEXT má pouze doplňkovou funkci a není při interpretaci programu nijak využíváno. Lze jej však použít pro načtení dalšího programu nebo libovolného jiného textového souboru. Otevřený soubor může mít délku maximálně 32 kB a je možné jej editovat stejným způsobem jako program. Okno lze použít například pro otevření dalšího programu, z důvodu převodu zapsaných textů přes schránku Windows. Další možností je například použití okna pro zápis dokumentace k programu a podobně.

3.0. Režimy práce systému OZOGAN KLONDAIK

Protože systém OZOGAN KLONDAIK je interpret jazyka a ve svém integrovaném prostředí obsahuje mimo editoru a interpretu další podpůrné prostředky, probíhá činnost systému v několika režimech. Každý režim odpovídá samostatné vývojové fázi vzniku, ladění a provozu programu. Každý režim má svá specifika s danými možnostmi práce se systémem.

Editační režim
Ladící režim
Režim animace
Provozní režim

3.1. Editační režim

Editační režim je nastaven automaticky vždy při spuštění systému OZOGAN KLONDAIK. Hlavním úkolem režimu je editace programu ve vnitřním jazyce INTER-PASCAL v okně 'program'. V editačním režimu lze mimo zápisu programu využívat i další okna systému. Výhodné je například použití okna 'příkazy', kdy lze zadávat jednotlivé příkazy, které jsou bez spuštění programu ihned interpretovány a provedeny. Tak je možné si jednoduše vyzkoušet činnost většiny zabudovaných procedur a funkcí. V editačním režimu jsou přístupné pouze proměnné, které byly globálně deklarovány při posledním spuštění editovaného programu.

Okno 'výstup-grafika' je možné v editačním režimu použít jako jednoduchý grafický editor. Kreslit je přitom možné nejen pomocí definovaných povelových tlačítek, ale také pomocí zabudovaných procedur a funkcí pro výstupní grafické okno zadávaných z příkazového okna.

3.2. Ladící režim

Ladící režim programu je zahájen volbou menu pro krokování programu (případně stiskem funkčních kláves F7 nebo F8). Do ladícího režimu přejde interpretovaný program také po stisku tlačítka STOP, případně provedením příkazu SUSPEND z interpretovaného programu.

V ladícím režimu je interpretace programu dočasně pozastavena a uživatel má možnost použít příkazové okno pro zobrazení stavu proměnných, změnu jejich hodnot, případě provedení požadovaných příkazů. Obsah proměnných je také zobrazován v okně 'sledování proměnných'. Jejich hodnoty jsou přitom aktualizovány po každém kroku interpretovaného programu v ladícím režimu. K dispozici je také výstupní textové i grafické okno, kdy je možné například vytisknout, případně vynulovat jejich obsah. Je dokonce možné i editovat program, provedené změny však budou účinné až při následujícím spuštění programu a v případě přidání či výmazu řádků programu nebude souhlasit zobrazení krokovaného programu ani animace programu !

3.3. Režim animace

V režimu animace, který je zahájen stiskem funkční klávesy F6 je program automaticky interpretován krokováním programu řádek po řádku s průběžným zobrazováním právě vykonávané řádky programu. V režimu animace není možné zadávat v příkazovém okně příkazy. Obsah požadovaných proměnných je možné zobrazit s průběžnou aktualizací v okně 'sledování proměnných'. Režim animace je tudíž skloubení provozního a ladícího režimu a má sloužit především pro zobrazení vnitřní činnosti interpretovaného programu.

3.4. Provozní režim

Po spuštění programu stiskem klávesy F9 je nastaven provozní režim systému. Další možností přechodu do provozního režimu je vyvolání volby menu 'Program/spustit'.

V provozním režimu je prováděna interpretace zdrojového textu programu v jazyce INTER-PASCAL. Provozní režim je ukončen buď provedením editovaného programu s návratem do editačního režimu, nebo přerušením interpretace s přechodem do ladícího režimu. Pokud je program ukončen z důvodu chyby v programu, přejde systém automaticky po oznámení místa a příčiny chyby do editačního režimu. Přerušení interpretace programu je možné provést stiskem tlačítka STOP, případně z volaného programu procedurou SUSPEND.

Pokud se nachází systém v provozním režimu, není možné používat uživatelsky příkazové okno ani grafické výstupní okno. Činnost grafického a textového výstupního okna je závislá na interpretovaném programu. V provozním režimu není obnovován stav proměnných v okně sledování proměnných.

4.0. MENU SYSTÉMU OZOGAN KLONDAIK

Činnost systému je řízena pomocí menu, které obsahuje přehledně všechny důležité akce. Menu je přístupné po stisku funkční klávesy F10. Většina voleb menu je spustitelná také přímo pomocí definovaných funkčních kláves. Některé často používané volby jsou přístupné také pomocí ikon v toolbaru, který je umístěn pod řádkou hlavního menu.

Soubor
Úpravy
Program
Okna
Pomoc

4.1.0. Soubor

Tato položka menu dovoluje otevřít nebo vytvořit nový program, uložit jej pod novým názvem nebo jej vytisknout. Obsahuje také volbu pro ukončení systému.

Nový program
Otevřít (Ctrl+O)
Uložit (Ctrl+S)
Uložit pod názvem
Konec

4.1.1. Nový program

Volba se používá pro založení nového programu. Pokud byl předchozí program editován a změny nebyly uloženy, bude nejprve vyvoláno dialogové okno pro uložení programu. Nový vygenerovaný program bude obsahovat hlavní proceduru Main.

4.1.2. Otevřít (Ctrl+O)

Umožní vám otevřít již existující program. V dialogovém okně máte možnost listovat v aktuálním adresáři, případně provést změnu pracovního adresáře nebo disku. Načte-li se program z disku, obsahuje záhlaví okna adresář a název editovaného programu.

4.1.3. Uložit (Ctrl+S)

Provede se uložení aktuálního stavu programu do souboru. To je výhodné pro průběžné ukládání provedených změn. Předchozí verze programu bude přitom zrušena, nevytváří se záložní kopie. Budete-li chtít uložit nový program, budete nejprve dotázáni na jeho název.

4.1.4. Uložit pod názvem

Používá se pro změnu jména editovaného programu. Program je možné uložit do jiného adresáře, případně na jiný disk. Původní program zůstane přitom zachován beze změny.

4.1.5. Konec

Bude ukončena činnost systému OZOGAN KLONDAIK. Pokud byly provedeny změny v editovaném programu, budete dotázáni na možnost uložení změn.

4.2.0. Úpravy

Menu obsahuje volby pro úpravu a editaci programů, vyhledávání a změnu textu.

Vyjmi (Ctrl+X)
Zkopíruj (Ctrl+C)
Vlož (Ctrl+V)
Vymaž
Hledat (Ctrl+F)
Změnit (Ctrl+L)

4.2.1. Vyjmi (Ctrl+X)

Označený text je přesunut z programu do schránky. Takto přesunutý text lze poté pomocí volby Úpravy/Vlož přesunout na jiné místo programu, případně do příkazového okna.

4.2.2. Zkopíruj (Ctrl+C)

Zkopíruje označený text z programu do schránky. Text zůstane v programu zachován. Zkopírovaný text lze pomocí volby Úpravy/Vlož zkopírovat na jiné místo v programu, případně použít v příkazovém okně.

4.2.3. Vlož (Ctrl+V)

Text, který byl překopírován, případně přesunut z programu je možné touto volbou přesunout na libovolné místo v programu. Text se přesune na místo, na kterém je kurzor. Obsah schránky zůstane přitom zachován beze změny.

4.2.4. Vymaž

Odstraní označený text z programu, ale neuloží jej do schránky. Text je možné opět obnovit ještě před provedením dalších změn v programu stiskem kláves Ctrl+Z. Proto je nutné používat této volby menu s velkou opatrností.

4.2.5. Hledat (Ctrl+F)

Vyvolá se standardní okno pro hledání zadaného textu. Mimo hledaného textu je možné je zadat, zda se mají hledat pouze celá slova, zda rozlišovat velká a malá písmena. Možné si je také zvolit směr hledání textu (směrem na konec nebo na začátek textu).

Text je možné hledat nejen v programu, ale také v příkazovém okně, případně textovém výstupním okně.

4.2.6. Změnit (Ctrl+L)

Volba menu umožní vyhledání zadaného textu s jeho případnou záměnou na jiný text. V dialogovém okně lze zadat hledaný text, text pro náhradu a zda se mají hledat pouze celá slova a jestli rozlišovat velká a malá písmena.

4.3.0. Program

Obsahuje volby menu pro spouštění a krokování programu.

Spustit (F9)
Skok přes procedury (F8)
Krokovat po řádcích (F7)
Spustit do pozice kurzoru (F4)
Animace programu (F6)
Ukázat aktuální řádek
Restart programu (Ctrl+F2)
STOP programu

4.3.1. Spustit (F9)

Pokud jste v editačním režimu, zahájí volba menu vykonávání instrukcí programu od jeho počátku. Pokud se nacházíte v ladícím režimu, pokračuje vykonávání instrukcí programu od místa přerušení. Spuštěním programu touto volbou se přepne systém do provozního režimu.

Činnost programu může být ukončena mimo běžného ukončení programu dokončením interpretace, případně ukončením programu z důvodu nalezení chyby i dalšími způsoby. Přímo z interpretovaného programu lze provést buď jeho ukončení procedurou Halt nebo přerušení procedurou Suspend s přechodem do ladícího režimu. Uživatelsky lze provést ukončení programu stiskem ikony STOP, kdy přejde systém do ladícího režimu, případně volbou menu Program/restart. V tomto případě je interpretace násilně ukončena s přechodem do editačního režimu.

Pokud bude ve spuštěném programu nalezena jakákoliv chyba, oznámí interpret místo nalezení chyby (číslo řádku) a přibližnou specifikaci chyby a vykonávání programu bude ukončeno s návratem do editačního režimu.

4.3.2. Skok přes procedury (F8)

Nachází-li se program v editačním režimu, provede se nejprve spuštění editovaného programu kdy proběhne inicializace proměnných, procedur a funkcí. Po provedení prvního řádku procedury Main přejde systém do ladícího režimu a opakovaným zadáváním volby je možné program krokovat s možností sledování vykonávání instrukcí programu.

Pokud se již program nachází v ladícím režimu, provede se pouze vykonání jednoho řádku editovaného programu. Je-li v řádku volání procedury nebo funkce, provede se volání této procedury (funkce) jako jeden příkaz. Volaná procedura nebo funkce se tedy nekrokuje. Toho lze s výhodou použít, pokud je již volaná procedura nebo funkce odladěna.

Krokování programu vám takto umožní sledovat přehledně průběh programu po jednotlivých řádcích a kontrolovat přitom volání procedur. Pokud budete mít v okně sledování proměnných zadány důležité proměnné, budete moci současně sledovat změnu jejich hodnot.

Mezi jednotlivými kroky je možné zadávat z příkazového okna příkazy, měnit hodnoty proměnných apod.

4.3.3 Krokovat po řádcích (F7)

Nachází-li se program v editačním režimu, provede se nejprve spuštění editovaného programu kdy proběhne inicializace proměnných, procedur a funkcí. Po provedení prvního řádku procedury Main přejde systém do ladícího režimu a opakovaným zadáváním volby je možné program krokovat s možností sledování interpretace instrukcí programu.

Pokud se již program nachází v ladícím režimu, provede se pouze vykonání jednoho řádku editovaného programu. Je-li v řádku volání procedury nebo funkce, bude krokování pokračovat v těle této procedury (funkce). Tak je možné přehledně krokovat průběh programu po jednotlivých řádcích a sledovat přitom volání procedur. Pokud budete mít v okně sledování proměnných zadány důležité proměnné, budete moci současně sledovat změnu jejich hodnot.

Mezi jednotlivými kroky je možné zadávat z příkazového okna příkazy, měnit hodnoty proměnných apod.

4.3.4. Spustit do pozice kurzoru (F4)

Nachází-li se program v editačním režimu, zahájí se vykonávání instrukcí programu od jeho počátku. Pokud dojde interpret na řádku, na které se nachází kurzor, řádka se již neprovede a systém přejde do ladícího režimu.

Pokud je při vyvolání volby systém v ladícím režimu, bude program pokračovat v činnosti bez přerušení až dokud nedojde interpret na řádku, na které se nachází kurzor.

Volba menu se používá v případě, že chcete sledovat krokování pouze určité části programu. Například pokud vás zajímá pouze provádění určité procedury. Nastavte proto kurzor na začátek této procedury a stiskněte F4. Po vstupu interpretu na tento řádek dojde nebude již tento vykonán a program se přepne do ladícího režimu.

4.3.5. Animace programu (F6)

Nachází-li se program v editačním režimu, provede se nejprve spuštění editovaného programu kdy proběhne inicializace proměnných, procedur a funkcí. Potom se provádí automaticky interpretace jednotlivých řádek programu s průběžným zobrazováním vykonávaného kódu programu.

Pokud se již program nachází v ladícím režimu, bude animace programu pokračovat od poslední provedené řádky programu. Animace probíhá automaticky, řádek po řádku včetně volání procedur a funkcí. Nahrazuje tím opakovanou volbu menu krokovat po řádcích. Animaci programu je možné ukončit kliknutím na tlačítko STOP, kdy přejde systém do ladícího režimu. Dále je potom možné buď pokračovat v animaci, případně v manuálním krokování programu nebo klávesou F9 spustit provozní režim systému.

4.3.6. Ukázat aktuální řádek

V ladícím režimu ukáže zvýrazněním aktuální řádek programu. Volbu je výhodné použít například když se při krokování programu přesunete kurzorem na jiné místo v programu, aby jste si například prohlédli deklaraci vyvolané procedury. Pokud není program v ladícím režimu, neprovede volba menu žádnou akci.

4.3.7. Restart programu (Ctrl+F2)

Pokud je program v ladícím režimu, bude proveden restart programu a tím k okamžitému ukončení vykonávání programu. Při následujícím spuštění programu bude zahájeno jeho vykonávání znovu od prvního řádku.

4.3.8. STOP programu

Přeruší interpretaci programu s přechodem do ladícího režimu.

4.4.0. Okna

Menu umožňuje ovládat okna systému, to je otevírat, zmenšovat a uspořádat na ploše.

Za sebou
Vedle sebe
Srovnat ikony
Minimalizovat

Příkazy
Program
Výstup-grafika
Výstup-text
Databáze
Tabulka
Grafy
Sledování proměnných
Texty

4.4.1. Za sebou

Provede rovnoměrné rozložení otevřených oken na ploše obrazovky tak, aby zabírala celou plochu. To je výhodné pro rychlý přehled o tom, co se ve kterém okně nachází.

4.4.2. Vedle sebe

Otevřená okna se zobrazí tak, že se stupňovitě překrývají. Ze všech oken je viditelný pouze horní řádek s titulkem okna. Poslední okno na popředí zabírá zbytek obrazovky.

4.4.3. Srovnat ikony

Srovná ikony minimalizovaných oken v dolní části obrazovky.

4.4.4. Minimalizovat

Provede minimalizaci všech otevřených oken do tvaru ikony a srovná je rovnoměrně v dolní části obrazovky.

4.4.5. Příkazy

Okno příkazy umožňuje zadávat kdykoliv většinu příkazů jazyka přímo z klávesnice bez nutnosti spuštění laděného programu. Tak je možné si vyzkoušet činnost příkazů, procedur a funkcí bez zápisu programu. Při spuštění programu v ladícím režimu je možné zadávanými příkazy vypisovat obsah proměnných, případně změnit jejich hodnoty. Příkazové okno je velmi výhodné zejména při prvních pokusech s programem, protože umožňuje ihned vykonávat (interpretovat) vaše příkazy bez nutnosti zápisu programu.

Podrobněji viz Okno příkazy

4.4.6. Program

Jedná se o nejdůležitější okno programu, ve kterém je možné editovat vyvíjený program. Program může mít délku maximálně 32 Kb, v textu programu je možné zabudovaným editorem vyhledávat požadovaný text a provádět nahrazování textu. Program je možné vytisknout. Okno s programem se používá i při ladění a krokování programu, kdy je v okně zobrazován aktuálně vykonávaný řádek programu.

Podrobněji viz Okno program

4.4.7. Výstup-grafika

Výstupní grafické okno představuje kombinaci jednoduchého grafického editoru pro zpracování bitmapových obrázků s možností kreslení obrazců pomocí příkazů z programu. Tak je možné například celý obrázek naprogramovat. Grafické okno podporuje množství příkazů jazyka pro kreslení, ale také tisk obrázku a výstup do souboru typu BMP.

Podrobněji viz Okno výstup-grafika

4.4.8. Výstup-text

Okno je určeno pro textové výstupy z programu příkazem WRITE. Možné je tak zobrazovat výsledky výpočtů, případně zprávy uživateli. Obsah okna lze vymazat buď příkazem z programu nebo kliknutím myši na příslušnou ikonu. Stejným způsobem je možné vytisknout obsah okna na tiskárnu. Možné je také uložit obsah okna do textového souboru.

Podrobněji viz Okno výstup-text

4.4.9. Databáze

Okno je se používá pro práci s databází ve formátu FoxPro. Možnosti práce s databází se Vám zobrazí po stisku pravého tlačítka myši na ploše databázového okna. Je možné nastavit možnost editace databáze v okně, nastavit filtr výběru záznamů, vybrat jednu ze tří pracovních oblastí a uzavřít databázi.

V demoverzi programu není možné vytvářet nové databáze, ani měnit strukturu existujících databází.

Podrobněji viz Databázové okno

4.4.10. Tabulka

V okně je možné zobrazit tabulku ve formátu Excel 4. S údaji v tabulce je možné provádět základní operace jako u běžných tabulkových procesorů. S tabulkou lze manipulovat i pomocí procedur a funkcí přimo z programu, případně z příkazového okna.

Podrobněji viz okno tabulka

4.4.11. Grafy

Okno pro zobrazování grafů s možností zadávání hodnot, nastavení typu a popisů grafu. Parametry a hodnoty grafu je možné zadávat i z programu. Graf je možné vytisknout.

Podrobněji viz okno grafy

4.4.12. Sledování proměnných

Pomocné okno je určeno pro průběžnou kontrolu obsahu použitých proměnných v programu v ladícím režimu. V případě potřeby je možné provést změnu nastavení hodnoty proměnné v příkazovém okně.

Podrobněji viz Okno sledování proměnných

4.4.13. Texty

Okno text je určeno pro editaci libovolného textového souboru. To je vhodné především pro současné otevření dvou programů, z důvodu převodu zapsaných textů přes schránku Windows. Stejně tak je možné v okně program ladit zdrojový text programu a v okně text současně psát libovolné poznámky, případně dokumentaci.

Podrobněji viz Okno text

4.5.0. Pomoc

Menu obsahuje podpůrné prostředky systému, to je nápovědu k programu a informace o autoru programu.

Help
O programu

4.5.1. Help

Volba vyvolá hypertextový dokument s celkovou nápovědí k programu. Nápověda obsahuje nejen popis obsluhy k programu, ale i popis vnitřního jazyka INTER-PACAL a popis použití systému a jazyka.

Větší část informací obsažených v nápovědě k programu je obsažena i v tištěných příručkách. Přesto však obsahuje nápověda podrobnější informace. V některých případech i aktuálnější. Značnou výhodou je možnost použití hypertextových odkazů na související informace.

4.5.2. O programu

Zobrazí informace o programu a jeho autorovi včetně kontaktní adresy.

5.0. KURZ POUŽÍVÁNÍ PROGRAMU

Pro ty, kteří dosud nikdy neprogramovali na počítači se může zdát pojem programování záhadný a tajemný. Jedná se však o zcela běžnou věc, kterou vlastně všichni známe již od dětských let. Určitě jste si již mnohokrát řekli: "ráno půjdu do školy (práce), odpoledne na koupaliště a večer se budu dívat na televizi". Tím jste si sestavili program dne. Definovali jste posloupnost akcí, které provedete. Můžete si však také stanovit podmínky programu. Například pokud bude odpoledne hezky, půjdete na koupaliště, jinak (bude pršet) si budete číst nebo se učit. V některých případech si můžete stanovit opakování některých akcí. Například že se budete učit tak dlouho, dokud to nepochopíte. Jak vidíte, programovat už umíte. Sestavujete si program dne, výuky, dovolené. Nyní už zbývá pouze naučit se převést program do podoby, kterou zvládne i počítač.

Počítačový program představuje definici posloupnosti akcí, podmínky jejich provedení a opakování. Aby byl počítač schopen požadované činnosti provádět, musíte mu je zadat v podobě, které on rozumí. Pokud budete chtít angličanovi říct svůj program dne, budete mu jej muset přeložit do angličtiny. Stejně tak musíte přeložit program ze slovní podoby do počítačového jazyka. Počítač potom bude číst jednu řádku programu za druhou a vykonávat to, co jste mu zadali.

V dalších lekcích bude popsáno, jakým způsobem budete s počítačem komunikovat, jak zapsat program. Naučíte se také základům použitého počítačového jazyka.

1.lekce - instalace a spuštění programu

Postup instalace programu záleží na verzi programu, kterou máte k dispozici a na použitém zdrojovém médiu. Demoverze, kterou máte možnost získat na internetu je šířena ve zkomprimovaném formátu v souboru KLONDAIK.ZIP a stačí ji pouze "rozbalit" do libovolného adresáře na disku počítače. V tomto případě se nezakládá programová skupina ve Windows a spuštění programu je nutné provést spuštěním souboru KLONDAIK.EXE. Demoverze zabírá na disku asi 1 Mb prostoru.

Demoverze šířená na disketách i plná verze programu obsahuje již instalační program. Program se dodává na jedné disketě formátu 3,5" HD. Instalace se spustí z diskety příkazem INSTALL.EXE. Zobrazí se vám řídící instalační panel, ve kterém musíte zadat nejprve adresář pro instalaci programu. Standardně je pro instalaci nastaven adresář C:\OZOKLOND\ pro demoverzi, případně C:\OZOKLON2 pro plnou verzi programu. Cílový disk i adresář je možné změnit, přesto však doporučujeme zachovat předdefinované jméno adresáře. Kopírování souborů do zadaného adresáře se zahájí po stisku tlačítka "Instalace". Program zabírá na disku asi 1 Mb prostoru. Po zkopírování souborů je nabídnuta možnost založení skupiny programů Windows. Pokud budete souhlasit, založí se skupina OZOGAN, do které se nadefinuje spuštění programu KLONDAIK, případně prohlížení helpu k programu.

Pro rozlišení přesné definice programu OZOGAN KLONDAIK od editovaného a laděného programu bude v následujícím textu u všech lekcí uváděn program OZOGAN KLONDAIK jako systém a pod pojmem program se bude rozumět laděný a editovaný uživatelův program.
Systém se startuje po dokončené instalaci spuštěním souboru OZOKLOND.EXE (demoverze), případně OZOKLON2.EXE (plná verze programu). Při prvním spuštění systému je integrované uživatelské prostředí nastaveno do standardního stavu.

Program je možné ukončit buď volbou z menu Soubor/Ukončit program, případně ikonou z toolbaru. Pokud je editován program a změny nebyly dosud uloženy, budete dotázáni systémem na uložení změn. Při ukončení programu se provede uložení rozložení oken integrovaného prostředí do inicializačního souboru OZOKLON*.INI (dle druhu verze programu), který se nachází v adresáři Windows. Při dalším spuštění programu bude potom obnoven stav rozložení jednotlivých oken na pracovní ploše systému.

2.lekce - základy ovládání programu

Systém OZOGAN KLONDAIK je aplikace Windows a proto je ovládání programu podřízeno tomuto standardu. Systém je ovládán z menu, často používané volby jsou přitom přístupné i z toolbaru pomocí ikon, případně pomocí funkčních kláves. Veškeré akce probíhají v několika oknech s přesně definovaným určením. Celá práce se systémem probíhá v několika režimech dle kterých jsou využívána příslušná okna. Okna lze na ploše obrazovky uspořádat podle aktuální potřeby. Okna je možné minimalizovat, nelze je však žádným způsobem zrušit.

Nejčastěji používané okno je okno s programem, které obsahuje vyvíjený a laděný program. V dalším, příkazovém okně je možné zadávat přímo příkazy jazyka bez nutnosti spuštění programu. Výsledky programu jsou zobrazovány buď v textovém nebo grafickém výstupním okně. Při krokování programu je možné zobrazovat stav proměnných v okně sledování proměnných. Pomocnou funkci má okno text, ve kterém je možné zobrazit libovolný textový soubor. Některá z oken mají svou vlastní sadu ikon umístěných v toolbaru, který se zobrazí po aktivaci okna v jeho horní liště.

Okna můžete po pracovní ploše libovolně přepínat, posouvat, zvětšovat je či zmenšovat. Přepínání oken je možné kliknutím na libovolnou viditelnou část okna. Po kliknutí se stane příslušné okno aktivní. Pokud není okno viditelné, použijte menu Okna, kde zvolíte požadované okno. V dalších lekcích se naučíte, že je možné zviditelnit požadované okno i příkazem z programu. Kliknutím na příslušnou ikonu v pravém horním rohu okna je možné okno maximalizovat, případně minimalizovat. Okno se přesouvá nejlépe myší, kdy je uchopíte za horní lištu a přesunete na požadované místo. Změny velikosti okna dosáhnete po uchopení pravého dolního rohu okna a nastavení na požadovanou velikost.

Vždy při spuštění systému je nastaven editační režim, ve kterém je možné editovat vytvářený program, případně zadávat příkazy z příkazového okna. Po spuštění editovaného programu je nastaven provozní režim, ve kterém je provozována laděná aplikace. Zvláštním druhem provozního režimu je potom ladící režim, ve kterém lze krokovat program a amimační režim, při kterém je možné zobrazovat průběh činnosti programu. Každý režim má svá specifika, která budou popsána v následujících lekcích.

Systém obsahuje kontextovou nápovědu, která je dostupná v několika stupních. Při výběru volby z menu je krátká nápověda k vybrané volbě zobrazována ve stavovém řádku v dolní části systému. Tzv. bublinová nápověda je zobrazována po najetí myší na ikony v toolbaru. Nejobsáhlejší nápověda se zobrazí po stisku klávesy F1. Jedná se o hypertextový dokument s provázanými odkazy na zvolená hesla. Zde uváděné příklady programů máte možnost si do systému z nápovědy přetáhnout pomocí blokového přesunu přes schránku Windows. To provedete tak, že najedete v nápovědě myší na začátek ukázky programu, stisknete levé tlačítko myši, držíte jej stisknuté a přetáhnete myš na konec textu příkladu. Tam tlačítko myši uvolníte. Stiskem kláves Ctrl+C se převede takto označený blok textu do schránky Windows. Nyní se přepnete do okna s programem a stiskem kláves Ctrl+V převedeme obsah schránky.

K ovládání programu je vhodné používat myš. Většina voleb je sice dostupná současně z menu i myší, přesto je však ovládání myší mnohem příjemnější a operativnější. Pokud se v dalším textu používá myš, předpokládá se levé tlačítko.

3.lekce - práce s příkazovým oknem

Nejprve si ukážeme, jak se dají velmi jednoduše zadávat systému příkazy bez zápisu programu. Najděte na pracovní ploše okno nazvané příkazyKLON1_21 a učiňte je aktivním - klikněte na něj myší. Pokud není okno viditelné , případně je minimalizované, můžete zvolit z menu volbu Okna/Příkazy nebo stiskem kláves Ctrl+F2. Upravte myší velikost okna tak, aby zabíralo levou polovinu obrazovky. Obdobným způsobem nastavte okno Výstup-text do pravé poloviny obrazovky. Přepněte ze zpět do příkazového okna.

Okno příkazy se podobá textovému editoru. Má však velmi důležitou vlastnost. Pokud napíšete v okně libovolný text a stisknete klávesu ENTER, pokusí se systém ihned napsaný řádek interpretovat jako známý příkaz. To znamená, že systém si přečte text řádku, na kterém stál kurzor a zjišťuje, zda se jedná o jemu známý povel nebo příkaz. Pokud ano, povel nebo příkaz provede. Proto se toto okno nazývá příkazové.

Příkazy a povely jsou přitom pro systém instrukce, co má vykonat. Příkazem je většinou označen přímý hlavní povel systému. Příkazy jsou v programech psány velkými písmeny. Jako povely jsou označovány doplňkové příkazy, které jsou zabudovány jako podpůrné akce. Jsou psány malými písmeny s velkými písmeny na počátku slova. Pokud jsou názvy povelů uvedeny z více slov, nesmí být mezi nimi uvedeny mezery.

Napište nyní v příkazovém okně slovo BEEP a stiskněte klávesu ENTER. Systém zjistí, že příkaz BEEP je pokyn, pro pípnutí. Zajistí proto, že reproduktor počítače krátce pípne. Takto systém interpretoval vámi zadaný příkaz do podoby srozumitelné pro počítač. Interpretace proto znamená převedení příkazu nebo řádku programu do formy srozumitelné pro zařízení počítače (obrazovka, tiskárna ...). Takto je v systému OZOGAN KLONDAIK interpretován každý řádek programu. Pokud je některý řádek několikrát opakován, provádí se i opakovaně jeho interpretace. Takovým systémům se proto říká interprety. Jiné systémy překládají zdrojový program pouze jednou přímo do spustitelné podoby. To jsou kompilátory a bývají rychlejší. Neumožňují ale interaktivní práci uživatele se systémem. Proto jsou pro začátky programování vhodnější interprety, které vám umožní lépe a rychleji proniknout do tajů programování.

V příkazovém okně máte možnost vyzkoušet si mnoho příkazů a povelů systému OZOGAN KLONDAIK, aniž napíšete jedinou řádku programu. Pokud uděláte chybu, systém vás na to upozorní a vy máte možnost ihned chybu opravit bez nutnosti kompilace programu. Zapomenete-li například napsat ve výše uvedeném příkazu BEEP jedno 'E', oznámí vám systém, že uvedený příkaz nezná. Najedete proto kurzorem na chybné místo, provedete opravu a po stisku klávesy ENTER se již příkaz vykoná.

Jistě jste si již všimli, že klávesa ENTER neukončí na místě kurzoru řádek, jak to bývá u textových editorů. Řádek zůstane celý zachován a kurzor se automaticky přesune na začátek prázdného řádku za předchozí příkazy. Pokud zkusíte zapsat příkaz

```
WRITELN("KLONDAIK");
```

vypíše se do textového výstupního okna text KLONDAIK. Možnostmi příkazu WRITELN je věnována následující lekce. Nyní bude stačit, pokud si zapamatujete, že příkaz WRITELN v uvedeném tvaru vypíše obsah textu uvedeného mezi uvozovkami do textového výstupního okna. Mezi uvozovky můžete zkusit přitom zadat libovolný jiný text. Zkuste text mezi uvozovkami několikrát změnit a změny vždy odešlete klávesou ENTER. Jistě si všimnete, že původní text před opravou zůstane v příkazovém okně zachován a příkaz s novým textem bude doplněn na konec příkazového okna. Díky tomu máte vždy přehled o historii zadávaných příkazů. Neprovedete-li v textu žádnou změnu a příkaz opakovaně odešlete, nebude stejný příkaz zadávaný beze změny za sebou v historii příkazového okna za sebou znovu opakován.

Historii zadávaných příkazů můžete libovolně používat bez ohledu na pořadí původního zadávání. Pokud provedete v některém řádku příkazového okna libovolnou změnu a následně stisknete klávesu ESC, příkaz se neprovede a text upraveného řádku bude uveden do původního stavu. Příkazy vyzkoušené v příkazovém okně máte možnost přes schránku Windows následně převést do programu.

Různými pokusy s příkazovým oknem jste dosáhli toho, že máte ve výstupním textovém okně uvedeny různé texty. S každým použitím příkazu WRITELN přibude na konec okna jeden řádek. To připomíná použití psacího stroje, kdy také není možné se vracet zpět. V dávných dobách prahistorie výpočetní techniky sálových počítačů se podobné psací stroje používaly pro komunikaci s počítačem. Tehdy se jim říkalo konzole - anglicky console. Uvedené anglické označení je proto také používáno pro příkazy přímo související s obsluhou textového výstupního okna. Pokud například zkusíte zadat v příkazovém okně příkaz ConsoleClear, zjistíte, že zruší obsah textového výstupního okna. Obdobného efektu dosáhnete, pokud se přepnete do textového výstupního okna a stisknete v podřízeném toolbaru uvedeného okna ikonu znázorňující prázdnou stránku. Po kontrolním dotazu bude obsah textového výstupního okna zrušen.

V následující kapitole se seznámíte, jak vypisovat v textovém výstupním okně za použití příkazu WRITELN i jiné informace a hodnoty.

4.lekce - příkaz WRITELN

V předchozí kapitole jste se naučili zadávat z příkazového okna příkazy umožňující vypsat do textového výstupního okna znaky. Nyní si ukážeme, jak vypisovat čísla a aritmetické výpočty.

S příkazem WRITELN jste již měli možnost se krátce seznámit. Používá se na výpis hodnot. Požadovaná hodnota se přitom uváděla v závorce. Pokud se u příkazu WRITELN uvedou pouze prázdné závorky, nebude nic vypsáno a další výstup do textového výstupního okna bude prováděn na novou řádku. WRITELN je pouze příkaz který uvádí systému, že má něco vypsat. Uvedené něco se přitom uvádí v závorce jako parametr příkazu. Pokud uvedete znaky v uvozovkách nebo apostrofech, převedou se uvedené znaky do textového výstupního okna. Pokud by jste chtěli vypsat číslo, musíte zadat příkaz v následujícím tvaru:

```
WRITELN(10);
```

Stačí tedy napsat jako parametr příkazu do závorek přímo požadované číslo. Číslo se vypíše se zarovnáním od levého kraje okna. Pokud budete chtít ponechat zleva odsazené místo, budete muset zadat požadovanou délku vypsaného čísla v počtech znaků:

```
WRITELN(10:5)
```

Délka vypsaného čísla je v tomto případě pět znaků. To je parametr uvedený v příkazu za dvojtečkou a udává počet míst zobrazení čísla. Nejedná se proto o výpočet dělení. Výpočty je ale samozřejmě také možné zadat. V tomto případě by jste pro dělení museli zadat:

```
WRITELN(10/5:5)
```

Jako výsledek bude zobrazeno číslo 2 o délce pěti znaků. Zkuste si sami zadat součet, rozdíl, případně násobek čísel:

```
WRITELN(10+5:5);
WRITELN(10-5:5);
WRITELN(10*5:5);
```

Zobrazí se vždy výsledek aritmetického výrazu o délce pěti znaků. Překvapeni budete možná pokud zadáte příkaz:

```
WRITELN(10/3:5);
```

Sami jistě poznáte, že něco není v pořádku. Ztratila se desetinná část výsledku. Až dosud jsme pracovali pouze s celými čísly. Výsledky se také vždy zobrazují standardně jako celá čísla. Zkuste nyní následující příkaz:

```
WRITELN(10/3:5:2);
```

Zobrazí se již výsledek s uvedením dvou desetinných míst. Je to proto, že jsme za dvojtečkou uvedli délku vypisovaného čísla a současně za další dvojtečkou počet zobrazených desetinných míst. Zkuste si sami další příklady a seznamte se zadáváním uvedených parametrů.

S možností výpisu nenumerických údajů, tedy znaků jste se již seznámili v předchozí lekci. Seskupení několika znaků se přitom nazývá řetězec a také tak již bude v následujícím textu uváděno. Pokud by jste chtěli jedním příkazem WRITELN vypsat najednou řetězec i znaky, můžete tak učinit uvedením několika parametrů oddělených čárkou:

```
WRITELN('Výsledek výpočtu je:', 10/3:5:2);
```

Stejně tak můžete vypsat samozřejmě i několik čísel nebo řetězců oddělených v příkazu čárkami. To vám dovolí ve spojení s možností uvádění délky čísel vhodnou grafickou úpravu textu:

```
WRITELN('Seznam výsledků:', 12/3:5:2, 45-26:5:2, 3*3:5:2);
```

Příkaz WRITELN provede po svém ukončení vždy přechod na novou řádku. Pokud však budete potřebovat, aby další výpis pokračoval na stejné řádce, můžete použít příkazu WRITE. Ten neprovádí ukončení řádku, následující výpis je zahájen od pozice ukončení příkazu WRITE. Možnosti a parametry příkazu WRITE jsou přitom totožné jako pro uváděný popis příkazu WRITELN.

Výše uvedený popis použití příkazu WRITELN se vám může zdát na první pohled složitý. Je však nutné jej přesně dodržet. Systém si hlídá důsledně jeho dodržování a v případě chyby odmítne příkaz vykonat. Přesná definice jakéhokoliv používaného jazyka se nazývá syntaktická pravidla. Pokud zadá uživatel chybný zápis, dojde k porušení syntaxe a odmítnutí systému k vykonání chybného zadání. Byla proto sestavena pravidla pro použití všech příkazů. Uvádíme zde pro představu zkrácenou definici syntaxe zápisu příkazu WRITELN tak, jak byla výše popsána:

```
WRITELN( [ výraz [:délka] [:desetiny] ] ) [;]
```

V zápise znamená výraz libovolný výraz, Například řetězec, číslo nebo výpočet. Údaje v hranatých závorkách se nemusí uvádět. Pokud proto uvedeme dvojtečku následovanou číslem, jedná se o délku vypisovaného čísla. Případná další dvojtečka s číslem uvádí počet desetinných míst. Pokud si dobře prohlédnete uvedený syntaktický zápis zjistíte, že mezi závorkami nemusíte nic uvádět. To je dáno levou hranatou závorkou uvedenou ihned za levou kulatou závorkou. Ukončení této volitelné části je provedeno pravou hranatou závorkou před pravou kulatou závorkou. Pokud proto uvedete příkaz:

```
WRITELN();
```

nevypíše se žádný text, dojde ale k vynechání prázdného řádku. Středník uvedený v hranatých závorkách na konci příkazu není nutné uvádět. V popisu syntaxe je uveden, protože v klasickém jazyku Pascal musí být každý řádek programu ukončen středníkem. Uvedený způsob popisu syntaxe je použit u popisu všech příkazů a knihoven procedur a funkcí uvedených v manuálu i helpu k programu.

Pokud jste se dokonale seznámili s použitím příkazu WRITELN, můžete ve spolupráci s manuálem k programu vyzkoušet některé funkce matematické knihovny. Dále uvedené příklady uvádějí na konci řádku mezi složenými závorkami komentáře, které nemají na výsledek žádný vliv. Ve skutečnosti je proto nemusíte včetně složených závorek uvádět.

```
WRITELN(Abs(-55));        {absolutní hodnota čísla}
WRITELN(Cos(PI));         {kosinus Ludolfova čísla PI}
WRITELN(Max(3,10);        {maximální hodnota zadaných čísel}
WRITELN(Min(3,10);        {minimální hodnota zadaných čísel}
WRITELN(Random(500);      {náhodné číslo do 500}
WRITELN(Round(12.82);     {zaokrouhluje číslo}
WRITELN(Sqr(5);           {vrací druhou mocninu čísla}
WRITELN(Sqrt(16);         {vrací druhou odmocninu čísla}
WRITELN(Trunc(12.82);     {odřízne desetinou část čísla}
```

Jak vidíte, můžete použít příkaz WRITELN jako docela chytrou kalkulačku. V současnosti při

použití pouze příkazového okna má ale jednu dosti velkou vadu. Nemá žádnou paměť, do které by jste si zaznamenali hodnoty mezivýsledků. V některé z dalších lekcí, kdy budeme probírat deklaraci proměnných v programu se ale naučíme i to.

V následujících lekcích se seznámíte s dalším výstupním oknem, které se používá pro výstup a kreslení grafiky.

5.lekce - grafické výstupní okno

V předchozích kapitolách jsme se naučili používat příkazové okno a textové výstupní okno. Systém ale obsahuje i další, velmi zajímavé výstupní okno. Je to grafické výstupní okno, které vám umožní do své plochy libovolně kreslit.

Uzavřete dříve používané textové výstupní okno. Můžete tak učinit kliknutím na uzavírací ikonu, případně můžete zadat v příkazovém okně povel ConsoleHide. Dále aktivujte grafické výstupní okno. To lze provést z menu volbou Okna/výstup grafika. Okno je možné také zobrazit povelem ImageShow. Zobrazené okno upravte opět tak, aby pokrývalo pravou polovinu obrazovky.

Jak vidíte, každé okno je pro účely ovládání z programu pojmenováno významově dle anglického názvosloví. Existují přitom povely pro zobrazení (anglicky show) a ukrytí (anglicky hide) okna. Možná se pozastavujete nad tím, že u českého programu se používá cizojazyčných termínů. Je to proto, že systém OZOGAN KLONDAIK může sloužit jako nástroj pro výuku programování. Proto je podle nás vhodné si již od počátku zvyknout na terminologii, která se při běžném programování používá. Názvy procedur, funkcí a konstant přitom vychází z používané terminologie jazyků Pascal a získané znalosti jistě dále zužitkujete. Použitá počítačová angličtina je přitom velmi jednoduchá a neměla by nikomu činit problémy.

Po aktivování grafického výstupního okna se vám v jeho horní liště zobrazí řada ikon pro ovládání zabudovaného grafického editoru a ikony pro možnost kreslení na grafické ploše. Ikony jsou umístěny do tří skupin. Levá skupina se používá pro načítání a ukládání obrázků v grafickém formátu *.BMP. Střední skupina slouží pro definici kresleného tvaru a v pravé skupině naleznete nastavení typu čáry a výplně ploch.

Nejprve se seznámíme s možností kreslení základních geometrických tvarů. Ve střední skupině ikon jsou seskupena tlačítka pro čáru, obdélník, kružnici a obdélník se zakulacenými hranami. Klikněte nejprve na tlačítko s čárou. Tím jste zadali, že budete chtít kreslit čáry. Uvidíte, že tlačítko zůstalo stisknuto. Přesuňte ukazatel myši na grafickou plochu. Pokud nyní na grafické ploše stisknete tlačítko myši, podržíte jej a přesunete na novou pozici, bude se kreslit čára z bodu stisku tlačítka do aktuální pozice ukazatele myši. Po uvolnění tlačítka zůstane čára zachována. Zkuste si nakreslit i vodorovné a svislé čáry. Obdobným způsobem lze na grafickou plochu kreslit obdélníky, kružnice a obdélníky se zakulacenými rohy. Pro zvolení nového tvaru musíte stisknout příslušné tlačítko na liště s ikonami.

Čáry se kreslí černou barvou a tenkou čárou. Pokud budete chtít změnit barvu čáry, nebo její tloušťku, stiskněte ikonu v pravé horní části okna s vyobrazením tužky. Zobrazí se vám další řada ikon pro zadávání barvy, typu a tloušťky čáry. Po opětovném stisku ikony s tužkou se nastavení skryje. Ikona tedy pracuje jako přepínací tlačítko. Klikněte si proto na tlačítko tak, aby jste měli zobrazeny ikony pro nastavení čar. Vlevo je umístěna tabulka barev, uprostřed tlačítka pro výběr typu čáry a vpravo můžete zadat tloušťku čáry. Barvy se vybírají kliknutím na požadovanou barvu. Vybraná barva je označena dvoupísmennou anglickou zkratkou barvy. Typ čáry můžete vybrat plnou čáru, tečkovanou, čárkovanou, čerchovanou nebo můžete posledním tlačítkem úplně zrušit. Tloušťka čáry se udává v bodech. Pro změnu tloušťky můžete zapsat do editačního boxu přímo novou hodnotu, nebo můžete použít pro nastavení šipek. Pokud bude nastavena tloušťka čáry větší než jedna, neuplatní se zadaný typ čáry a čára se bude vykreslovat vždy plná, případně se nebude kreslit vůbec. Zkuste si nastavit parametry čáry a prověřte si účinky změny při kreslení základních geometrických tvarů.

Dosud jsme kreslili pouze okraje geometrických tvarů. Standardně je totiž nastaveno, že se plocha kreslených geometrických tvarů nevykresluje. Podobným způsobem, jako se nastavují parametry kreslení čar máte možnost nastavit parametry vykreslování ploch. Ikony pro nastavení se zobrazí po stisku tlačítka s vyobrazením štětce. Opět máte možnost zadat barvu, tentokrát plochy (výplně).

Současně můžete zadat typ výplně. Máte možnost vybrat si buď kreslení plné plochy zadanou barvou, nevykreslování plochy, nebo z několika druhů vykreslení plochy čárami. Lze vybrat čáry vodorovné, svislé, vodorovné i svislé současně (mřížka) a různé druhy diagonálních čar. Tloušťka čáry výplně je přitom vždy jeden bod.

Nyní jste již schopni nakreslit na grafické ploše pouze za použití myši jednoduché obrázky. Vyzkoušejte si různé možnosti nastavení čáry a plochy. Parametry nastavení čar a plochy je možné změnit samozřejmě také přímo z programu, případně zadat z příkazového okna. To si však ukážeme až v následujících lekcích. Nyní bude pro vás jistě zajímavá možnost uložení nakreslených obrázků na disk do souboru pro pozdější použití a zpětná možnost načtení obrázku z disku do grafického okna pro provedení úprav. Ukážeme si také, jak je možné obrázek vytisknout.

Pokud máte grafickou plochu zaplněnou předchozími pokusy, můžete provést její výmaz pomocí ikony s obrázkem prázdné stránky. Ikona se nachází v levé skupině ikon umístěné v grafickém výstupním okně. Po kontrolním dotazu bude grafická plocha vymazána. Stejného efektu lze dosáhnout povelem ImageClear zadaným v příkazovém okně. Povel se zadává bez parametrů, v tomto případě se provede výmaz grafické plochy již bez kontrolního dotazu.

Nakreslené obrázky máte samozřejmě možnost uložit do souboru. Používá se známý a běžný bitmapový soubor typu *.BMP. Pro uložení stiskněte ikonu s obrázkem diskety. V dialogovém okně zadejte adresář a jméno souboru. Obrázek můžete uložit samozřejmě i povelem ImageSave z příkazového okna. Například:

```
ImageSave("obrazek.bmp")
```

Obrázky můžete i zpět načíst. Použijte ikonu s obrázkem šipky směřující do stránky. V dialogovém okně vyberte adresář a jméno souboru typu *.BMP. Načíst můžete i obrázky vytvořené v jiných grafických systémech. Po načtení obrázku je velikost grafické plochy nastavena dle načítaného obrázku. Obrázek můžete načíst i povelem z příkazového okna ImageLoad. Například:

```
ImageLoad("obrazek.bmp")
```

Pokud by jste chtěli vytvořený obrázek vytisknout, můžete tak učinit pomocí ikony zobrazující tiskárnu. Pro tisk obrázku z příkazového okna můžete použít povel ImagePrint.

V následující lekci se seznámíte s možnostmi ovládání grafického výstupního okna pomocí povelů z příkazového okna.

6.lekce - ovládání grafického okna

Předchozí lekce vás seznámila krátce se základními možnostmi použití grafického výstupního okna. Naučili jste se kreslit do grafické plochy pomocí myši, ukládat, načítat a vytisknout vytvořený obrázek. Nyní se seznámíte s dalšími možnostmi grafického okna a s parametry okna, které budete potřebovat pro programování grafických povelů. Poznáte, že vše, co bylo možné nastavit v grafickém okně budete mít možnost zadat pomocí povelu z příkazového okna.

Činnost grafického okna je možné si představit jako malířské plátno definovaných rozměrů, na které se kreslí perem (anglicky pen). Větší plochy je možné vybarvit štětcem (anglicky brush). Kreslí se přitom vždy nastavenou barvou. Systém obsahuje povely pro nastavení parametrů pera i štětce. S možnostmi se seznámíte v následujícím textu.

Při kreslení obrazců do grafického okna se kreslí obrazce čárou, jejíž typ je definován povelem ImagePenStyle a tloušťka čáry je definována povelem ImagePenWidth. Plocha nakreslených geometrických obrazců je vyplněna stylem zadaným povelem ImageBrushStyle.

Nejjednodušší je změna tloušťky čáry, kdy uvádíte přímo jako parametr povelu tloušťku čáry v bodech. Pro změnu tloušťky čáry na pět bodů použijete z příkazového okna následující povel:

```
ImagePenWidth(5);
```

Všimněte si, že pokud nastavíte sílu čáry povelem z příkazového okna, použije se nastavená síla čáry i pro následné kreslení pomocí myši přímo v grafickém okně. Obdobná vlastnost je platná i pro nastavení všech parametrů grafického okna. Máte proto možnost libovolně kombinovat zadávaní povelů z příkazového okna nebo jejich nastavení pomocí ikon. V budoucnu budete mít samozřejmě možnost uvedené parametry nastavit i přímo z programu.

Styl čáry máte možnost zadávat povelem ImagePenStyle. Předdefinováno je šest stylů. Jako parametr povelu musíte přitom zadat definovaný styl čáry. Parametr můžete uvést buď číselnou hodnotou, nebo jménem konstanty dle následující tabulky:

hodnota	konstanta	název stylu
1	psSolid	souvislá čára
2	psDash	přerušovaná čára
3	psDot	tečkovaná čára
4	psDashDot	čerchovaná čára
5	psDashDotDot	čerchovaná čára se dvěma tečkami
6	psClear	neviditelná čára

Příklady použití definice stylu čáry (text mezi složenými závorkami nemusíte psát, jedná se o poznámku):

```
ImagePenStyle(1);          { souvislá plná čára }
ImagePenStyle(3);          { tečkovaná čára }
ImagePenStyle(6);          { neviditelná čára }

ImagePenStyle(psSolid);    { souvislá plná čára }
ImagePenStyle(psDot);      { tečkovaná čára }
ImagePenStyle(psClear);    { neviditelná čára }
```

Existuje samozřejmě i způsob nastavení barvy čáry. Nejprve se však budeme muset seznámit s možnostmi použití barev v počítači. Zobrazování barev závisí na vlastnostech videokarty ve vašem počítači. Používáte-li barevnou VGA kartu, máte možnost zobrazit minimálně 16 barev. Po příslušném nastavení videoadaptéru je možné běžně zobrazovat 256 barev, výjimečně i více. Vy budete mít možnost nastavit v systému libovolnou barvu z rozsahu 16 miliónů barev. Skutečně zobrazená barva ale závisí na vlastnostech technického zařízení, protože se zobrazí vždy barva nejbližší.

Hodnotu barev je možné zadávat dvěma způsoby, které lze v programu libovolně kombinovat. Pokud budete používat pouze základní, šestnáctibarevnou paletu, můžete tak učinit zadáváním předdefinované konstanty udávající anglické jméno barvy. Stejnou barvu máte možnost zadat i pomocí tzv. RGB hodnoty.

Hodnota barev zadávaná definicí RGB znamená, že každá barva je definována jako poměr kombinace barev modré, zelené a červené. Pro každou barvu je možné volit hodnoty v rozsahu 0 až 256. Násobek těchto hodnot (modrá x zelená x červená) udává výslednou barvu. Výhodné je používat tzv. hexadecimálního zápisu, kdy jsou pro každou barvu vyhrazeny dvě pozice čísla s hodnotami od 00 (číslo 0) až do FF (číslo 256). Při použití hexadecimálního čísla je nutné uvést před číslem rozlišovací znak $. Viz tabulka hodnot barev.

```
hodnota   konstanta    název barvy
-----------------------------------------
$000000   clBlack      černá
$000080   clMaroon     kaštanově červená
$0000FF   clRed        světle červená
$008000   clGreen      tmavě zelená
$008080   clOlive      tmavě žlutá
$00FF00   clLime       světle zelená
$00FFFF   clYelow      žlutá
$800000   clNavy       tmavě modrá
$800080   clPurple     tmavě fialová
$808000   clTeal       tmavě modrozelená
$808080   clDkGray     tmavošedá
$C0C0C0   clLtGray     světle šedá
$FF0000   clBlue       modrá
$FF00FF   clFuschsia   fialová
$FFFF00   clAgua       modrozelená
$FFFFFF   clWhite      bílá
```

Příklady použití definice barvy čáry (text mezi složenými závorkami nemusíte psát, jedná se o poznámku):

```
ImagePenColor(clWhite);     { bílá barva čáry }
ImagePenColor(clRed);       { světle červená barva čáry }
ImagePenColor(clBlue);      { modrá barva čáry }

ImagePenColor($FFFFFF);     { bílá barva čáry }
ImagePenColor($0000FF);     { světle červená barva čáry }
ImagePenColor($FF0000);     { modrá barva čáry }
```

Podobně, jako je možné nastavit parametry kreslené čáry je možné nastavit parametry vykreslovaných ploch geometrických obrazců. Barva plochy se přitom zadává povelem ImageBryshColor. Používá se přitom výše zadaných hodnot a konstant pro definici parametru barvy. Příklady použití definice barvy plochy (text mezi složenými závorkami nemusíte psát, jedná se o

poznámku):

```
ImageBrushColor(clWhite);     { bílá barva plochy }
ImageBrushColor(clRed);       { světle červená barva plochy }
ImageBrushColor(clBlue);      { modrá barva plochy }

ImageBrushColor($FFFFFF);     { bílá barva plochy }
ImageBrushColor($0000FF);     { světle červená barva plochy }
ImageBrushColor($FF0000);     { modrá barva plochy }
```

Pro nastavení stylu vyplňování ploch se používá povel ImageBrushStyle, kde se jako parametr povelu udává buď číslem definovaný styl, nebo jméno konstanty dle následující tabulky:

```
-------------------------------------------------------------
hodnota   konstanta      název stylu
-------------------------------------------------------------
   1      bsSolid        vyplní oblast jednou barvou
   2      bsClear        vyplní oblast barvou pozadí
   3      bsHorizontal   vyplní oblast vodorovnými čárami
   4      bsVertical     vyplní oblast svislými čárami
   5      bsFDiagonal    diagonální čáry \\\\\\\
   6      bsBDiagonal    diagonální čáry ///////
   7      bsCros         vodorovné a svislé čáry
   8      bsDiagCross    vodorovné a svislé čáry diagonálně
-------------------------------------------------------------
```

Příklady použití definice stylu plochy (text mezi složenými závorkami nemusíte psát, jedná se o poznámku):

```
ImageBrushStyle(1);           {plné vybarvení plochy}
ImageBrushStyle(3);           {výplň vodorovnými čárami}
ImageBrushStyle(8);           {výplň diagonálními čárami}

ImageBrushStyle(bsSolid);     {plné vybarvení plochy}
ImageBrushStyle(bsHorizontal);{výplň vodorovnými čárami}
ImageBrushStyle(bsDiagCros);  {výplň diagonálními čárami}
```

Aby bylo možné kreslit v grafickém okně do přesně určených pozic, musí být zadány souřadnice pro kreslení. Souřadnice určují polohu jednotlivých bodů kresby. Souřadnice znamená, že musíte uvést vzdálenost v bodech od levého okraje grafického okna a vzdálenost v bodech od horního okraje grafického okna. Souřadnicový systém je tedy vztažen k levému hornímu rohu, který má souřadnici 0,0. Hodnoty ve směru osy X narůstají směrem doprava, hodnoty ve směru osy Y narůstají směrem dolů. Při zápisu souřadnice se uvádí nejprve osa x, potom osa y. Je přitom možné zadávat příkazy pro kreslení mimo plochu grafického okna, zobrazí se však pouze ta část, která je obsažena maximálními souřadnicemi grafického okna.

Aktuální souřadnice se zobrazují ve stavovém řádku systému vždy, když máte nastavenou myš nad grafickým oknem. Souřadnice je vhodné si vyzkoušet také na povelu Point, který slouží pro zobrazení bodu na zadané souřadnici. Povel nakreslí na zadaných souřadnicích bod o zadané velikosti. Bod se nakreslí aktuální barvou pera, kterou lze nastavit procedurou ImagePenColor. Vyzkoušejte si několik příkladů pro seznámení se se způsobem označování souřadnic grafické polohy:

```
Point(  0,   0,  2);    { levý  horní roh }
Point(  0, 100,  2);    { levý  dolní roh }
Point(100,   0,  2);    { pravý horní roh }
Point(100, 100,  2);    { pravý dolní roh }
Point( 50,  50, 10);    { uprostřed, větší bod }
```

Souřadnice grafického okna se použijí i pro kreslení geometrických tvarů pomocí povelů. Možné je kreslit čáru povelem Line, obdélník povelem Rectangle, kružnici nebo elipsu povelem Ellipse a obdélník se zaoblenými rohy povelem RoundRect. Povelem Triangle je možné nakreslit trojúhelník, což není pomocí myši možné. Pomocí povelů je možné také kreslit povelem Arc část křivky a povelem Pie kruhovou výseč. Vyzkoušejte si kreslení základních geometrických tvarů:

```
Line(20, 20, 50, 100);          { nakreslí čáru }
Rectangle(10, 10, 100, 100);    { nakreslí čtverec }
Ellipse(30, 30, 120, 120);      { nakreslí kružnici }
Ellipse(30, 30, 120, 60);       { nakreslí elipsu }
Triangle(10,100,55,10,100,100}; { nakreslí trojúhelník }

Arc(0,0,100,100, 50,0,0,50);    {levý  horní čtvrtkruh}
Pie(0,0,100,100, 50,100,100,50) {pravý dolní čtvrtkruh}
```

7.lekce - výstup do grafického okna

V předchozí lekci jsme se naučili kreslit do grafického výstupního okna z příkazového okna a nastavovat parametry čar a ploch. Nyní se zaměříme na nastavení grafické plochy a převod části grafického okna ze zásobníku Windows a zpět. Seznámíme se také s možností zápisu textu do grafického okna.

Až dosud jsme měli velikost grafické plochy nastavenou vždy podle toho, jak veliké bylo grafické okno v okamžiku aktivace grafické plochy. Další změnou velikosti grafického výstupního okna se již velikost grafické plochy neměnila. Pokud proto potřebujete nastavit velikost grafické plochy na požadované rozměry, můžete tak učinit povelem ImageInit. Povel inicializuje grafické okno a nastaví jeho velikost na rozměry zadané parametry. Nastaví současně bílou barvu plochy, styl štětce pro plně vybarvené plochy, černou barvu pera, sílu čáry na jeden bod. Inicializací se provede výmaz původního obsahu grafického okna. Vyzkoušejte si například následující hodnoty:

```
ImageInit( 50, 200);
ImageInit(120,  60);
```

Pokud potřebujete znát parametry již inicializované grafické plochy, můžete použít dotazu GetMaxX a GetMaxY, který vrací velikost grafické plochy zadané strany. Pokud budete například potřebovat vykreslit bod uprostřed grafické plochy bez ohledu na velikost aktuálně inicializované plochy, můžete tak učinit povelem:

```
Point(GetMaxX/2, GetMaxY/2, 5);
```

V některých případech by bylo vhodné kreslit čáry do grafické plochy zadáváním v absolutních přírůstkových hodnotách místo přesné definice souřadnic. Proto je v grafickém okně definován tzv. grafický ukazatel, který zaznamenává pozici vykreslení posledního bodu. Poloha grafického ukazatele se při použití některých povelů automaticky mění. Je možné ji nastavit i z programu povelem MoveTo. Čáry je možné potom zadat povelem LineTo definicí konečného bodu. Využitím uvedených povelů je možné například nakreslit libovolný mnohostranný mnohoúhelník. Vyzkoušejte si následující povely, které by měly v grafické ploše nakreslit čtyřúhelník:

```
MoveTo( 10,  10);   { přesun na počáteční bod }
LineTo(100,  10);   { horní  hrana }
LineTo(100, 100);   { pravá  hrana }
LineTo(100,  10);   { spodní hrana }
LineTo( 10,  10);   { levá   strana }
```

Až dosud jsme důsledně dodržovali, že je možné textové informace vypisovat zásadně do textového výstupního okna a grafické informace do grafického výstupního okna. Je však možné provádět výstup textových informací i do grafického okna. Nelze přitom použít povelu WRITELN, se kterým jsme se již dříve seznámili. Zobrazovat lze pouze textové informace. Čísla je nutné předem převést na řetězce. K výstupu textových informací se používá povelu TextOut. Jako parametry povelu se udávají souřadnice pro zobrazení textu v grafickém okně a řetězec, který se má zobrazit. Povel si můžete vyzkoušet například následujícími příklady:

```
TextOut(10, 10, "OZOGAN");
TextOut(10, 30, "KLONDAIK");
```

Text se při použití povelu TextOut vypisuje předdefinovaným fontem. Jeho změna se provede buď povelem ImageSetFont, kdy je možné zadat v dialogovém okně nejen jméno fontu, ale přímo i jeho styl a velikost. Další možností je použití ikony z grafického výstupního okna. Pokud

potřebujete nastavit pouze některé atributy fontu, můžete tak učinit povely ImageFontColor (barva fontu), ImageFontName (jméno fontu), ImageFontSize (velikost fontu) a ImageFontStyle (styl fontu).

Při nastavení barvy fontu se zadává u povelu ImageFontColorIMAGEFONTCOLOR jako parametr buď číslo barvy, nebo jméno konstanty udávající barvu. Tabulka hodnot barev je stejná jako pro nastavení barvy čar a ploch. U povelu ImageFontName, který nastavuje jméno fontu se uvádí přímo jméno fontu. Pokud není zadané jméno v systému Windows dostupné, použije se font s podobným jménem. Velikost fontu se uvádí v povelu ImageFontSize přímo požadovanou hodnotou. Příklad možných nastavení si prozkoušejte včetně výpisu textu po každé změně parametrů fontu:

```
ImageFontColor(clRed);     { nastaví červenou barvu fontu }
ImageFontColor($00FFFF);   { nastaví žlutou barvu fontu }
ImageFontName("Arial CE");{ nastaví font Arial CE }
ImageFontSize(12);         { nastaví velikost fontu 12 bodů }
```

Pro nastavení stylu fontu se v povelu ImafeFontStyle požívají jako parametr předdefinované hodnoty jednotlivých stylů. Uvádí se součet hodnot požadovaného výsledného stylu:

hodnota	konstanta	popis stylu
0	fsNormal	normální písmo
1	fsBold	tučné písmo
2	fsItalic	nakloněné písmo
4	fsUnderline	podtržené písmo
8	fsStrikeOut	přeškrtnuté písmo

Pro nastavení nakloněného podtrženého písma můžete zadat jednu z následujících možností:

```
ImageFontStyle(2 + 4);                    {součet hodnot}
ImageFontStyle(6);                        {součet 2 + 4}
ImageFontStyle(fsItalic + fsUnderline); {uvedení konstanty}
```

Pokud budete potřebovat převést nakreslený obrázek přes schránku Windows (clipboard) do jiné aplikace, můžete tak provést pomocí povelu InageToClip, kdy uvedete jako parametry souřadnice ohraničené plochy pro převod. Pokud budete chtít například přesunout do schránky obsah celého grafického okna, zadejte příkaz:

```
ImageToClip(0,0,GetMaxX,GetMaxY);
```

Načtení obsahu schránky se provádí povelem ImageFromClip, kdy se jako parametry uvedou souřadnice ohraničené plochy, kam se má obsah schránky převést. Pokud budete chtít převést obsah schránky Windows na celou plochu grafického okna, zadejte příkaz:

```
ImageFromClip(0,0,GetMaxX,GetMaxY);
```

Někoho z vás již určitě napadlo, že výše uvedené povely, pomocí kterých můžete převádět obrázky do schránky a načítat zpět ze schránky by bylo možné použít na přesun části obrázku na nové místo. Na to je ale výhodnější použít samostatného povelu ImageMove, který provede obě akce najednou. Zadáváte přitom jako parametry souřadnice místa, odkud se má načtení provést a souřadnice, kam se má plocha přenést. Neodpovídá-li přitom poměr stran zdrojové a cílové plochy, nebude zkopírovaný obraz uříznut, ale bude upraven (deformován) do zadaných

souřadnic. To je možné využít k mnoha zajímavým efektům. Například ke zvětšení části plochy apod.

Někteří jste již možná netrpěliví, kdy se začne programovat. Místo programování jsme zadávali pouze povely do příkazového okna. Tím jsme se vás snažili naučit nejprve používat několik základních povelů programovacího jazyka. Pokud by jsme začali ihned se základy programování včetně popisu použití povelů bylo by toho najednou moc. Proto jsme si nejprve probrali základy používání povelů a příkazů jazyka a v příští kapitole již začneme opravdu programovat.

8.lekce - první program, procedura main

V předchozích lekcích jsme si probrali základní informace o používání příkazového okna pro zadávání povelů jazyka přímo z klávesnice. Výstup výsledků byl prováděn do textového nebo grafického okna. V následujících lekcích využijeme získané poznatky již přímo při vývoji vlastních programů.

Programy se zadávají (zapisují) do samostatného okna program, které slouží jako textový editor programu. Pokud je okno na obrazovce viditelné, můžete jej aktivovat kliknutím myši na jeho plochu. V opačném případě využijte z menu volbu Okna/Program. Pokud budete chtít použít povelu z příkazového okna, budete muset zadat povel ProgramShow. Pro možnost automatického vytvoření základní kostry programu stiskněte klávesy Ctrl+N, nebo zadejte volbu menu Soubor/Nový. Do okna program se vygeneruje kostra prázdného programu.

Nový, prázdný vygenerovaný program je možné ihned spustit volbou z menu Program/Spustit, případně stiskem funkční klávesy F9 nebo kliknutím myši na ikoně s obrázkem běžícího panáčka. Prázdný program obsahuje pouze základní kostru bez žádného povelu, který by vykonával viditelnou činnost a proto program skončí, aniž by cokoliv učinil. Využijeme proto získaných zkušeností a doplníme text programu o nám již známé povely a příkazy. Dávejte si pozor na to, aby jste ponechali beze změny první a poslední řádek programu. Pokud by se tak nestalo, program by se nemusel vůbec spustit. Proveďte proto pouze doplnění programu o příkaz WRITELN dle následující ukázky:

```
PROCEDURE main
   // zde zapište váš program
   WRITELN("Ahoj, zdraví Vás systém KLONDAIK");
ENDPROC
```

Program spustíme funkční klávesou F9 a pokud máte viditelný obsah textového výstupního okna, zobrazí se v něm pozdrav od systému. Vyzkoušejte si další, dříve probrané povely, které zapisujte pouze do části programu mezi slovy PROCEDURE a ENDPROC. Takto by například mohla být upravena procedura main:

```
PROCEDURE main
   ConsoleShow;       { zobrazí a aktivuje textový výstup }
   ConsoleClear;      { vymaže plochu textového výstupu }
   WRITELN("Ahoj, zdraví Vás systém KLONDAIK");
ENDPROC
```

Obdobným způsobem můžete zadat i povely pro použití grafického výstupního okna. Opět použijeme pouze dříve probrané povely, které zapíšete do části programu mezi slovy PROCEDURE a ENDPROC. Text ve složených závorkách nemusíte zapisovat, jsou to poznámky, které nemají na činnost programu vliv. O tom, jak přebírat dále uvedené příklady z nápovědě k programu se dozvíte podrobně ve druhé lekci.

```
PROCEDURE main
   ImageShow;                    {aktivuje textový výstup}
   ImageInit(150, 200);          {inicializuje plochu}
   Rectangle(10, 10, 140, 190);  {nakreslí obdélník }
   ImageFontColor(clBlue);       {nastaví modrou barvu fontu}
   ImageFontSize(16);            {nastaví velikost fontu}
   TextOut(50, 30, "KLONDAIK");  { vypíše text }
ENDPROC
```

Jak jsme již hned v úvodu probíraných lekcí uvedli, je základem programování zadání posloupností akcí. To si můžete nyní vyzkoušet. Projděte si předchozí lekce a pokuste se z již získaných znalostí sestavit program, který například nakreslí dům a vedle něj zelený strom. Aby jste to neměli tak složité, uvádíme zde část programu pro nakreslení domu. Povely pro nakreslení strumu již doplňte sami. Samozřejmě můžete v programu provést i další úpravy dle vlastního uvážení. Aby jste se také naučili něco nového z ovládání uživatelského prostředí systému, zkuste tentokrát spustit program klávesou F6. Tomu, co uvidíte se říká animace programu. Podrobněji se s ní seznámíme až v některé z dalších lekcí. Nyní krátce jen to, že animace mimo interpretace programu současně v okně s programem souběžně zobrazuje interpretovaný řádek programu.

```
PROCEDURE main
   ImageShow;                        {aktivace grafické plochy}
   ImageInit(200, 150);              {inicializace graf.plochy}
   ImageBrushColor(clLime);          {nastavení zelené barvy}
   Rectangle(0, 130, 200, 150);      {nakreslení zahrádky}
   ImageBrushColor(clOlive);         {nastavení tmavě zelené}
   Rectangle(20,60, 105, 130);       {nakreslení stěny domu}
   ImageBrushColor(clYelow);         {nastavení žluté barvy}
   Rectangle(28, 70, 48, 90);        {horní levé okno}
   Rectangle(72, 70, 52, 90);        {horní střední okno}
   Rectangle(96, 70, 76, 90);        {horní pravé okno}
   Rectangle(28, 100, 48, 120);      {levé dolní okno}
   Rectangle(72, 100, 52, 130);      {dveře}
   Rectangle(96, 100, 76, 120);      {pravé dolní okno}
   Ellipse(140,20, 170, 50);         {sluníčko}
   ImageBrushColor(clRed);           {nastavení červené barvy}
   Triangle(20,60, 60,25, 105,60);{střecha}
ENDPROC
```

Takže jak vypadá program již víte. Měli by jste být již také schopni sestavit jednoduchý program z posloupnosti vykonávaných povelů. Zatím jsme si ale nic neřekli o tom, co znamená první a poslední řádek programu, ve kterých jsou uvedena klíčová slova PROCEDURE a ENDPROC. To se dozvíte v některé z následujících lekcích, ve které si vysvětlíme, co všechno program může a musí obsahovat a jakou má mít strukturu.

9.lekce - editace programu

Při zápisu programu se zapisuje text programu textovým editorem. Jedná se o tzv. programátorský editor, který neumožňuje nastavovat proměnný druh ani velikost písma. Není to ani nutné, protože zapsaný text programu slouží pouze jako zápis posloupnosti instrukcí pro systém interpretu.

Zápis nového programu zahájíte stiskem kláves Ctrl+N nebo volbou z menu Soubor/Nový. Do okna program se vytvoří nová prázdná struktura programu, kterou můžete běžným způsobem doplňovat o nové řádky programu. Již existující program můžete otevřít stiskem kláves Ctrl+O, nebo volbou z menu Soubor/Otevřít. Program máte možnost uložit Stiskem kláves Ctrl+S nebo volbou z menu Soubor/Uložit. Pokud se jedná o nový program, musíte v dialogovém okně zadat jméno souboru a tím i programu pro uložení. Pokud již zadaný soubor existoval, bude přepsán novou verzí. Neprovádí se přitom záložní kopie původního souboru a tím samozřejmě i programu !

Pokud budete chtít uložit program pod novým jménem, můžete tak učinit volbou z menu Soubor/Uložit pod jménem. Budete dotázáni na jméno nového souboru, do kterého se program uloží. Původní soubor zůstane přitom zachován beze změny. Při práci se soubory a tudíž i s programy vždy platí, že může být v jednom okamžiku aktivní vždy pouze jeden. Nemůžete editovat najednou několik programů.

Základy editace programu jsou obdobné, jako u všech textových editorů. Jakým způsobem zapsat nebo přepsat text snad není nutné uvádět. Stejně tak jak se dostat na konec programu, procházet text pořádcích a podobně. Seznámíme se ale s blokovými operacemi, které jsou při programování velmi výhodné. Pokud budete potřebovat přesunout část textu na jiné místo, budete si muset požadovaný blok textu nejprve označit. To můžete provést buď myší, kdy se přesunete na začátek bloku, stisknete tlačítko myši, držíte je stisknuté a přesunete ukazatel myši na konec požadovaného bloku textu. Text zapsaný v bloku se přitom zvýrazní. Z klávesnice můžete blok textu označit tak, že najedete kurzorem na požadovaný počátek bloku textu, stisknete klávesu Alt, podržíte ji stisknutou a přejedete kurzorem na požadovaný konec bloku textu. Po uvolnění klávesy Alt máte označen blok textu pro další operace. Pokud j budete chtít blok textu pouze vymazat, stiskněte nyní klávesu Delete. Označený blok textu bude zrušen. Pozor proto při práci s označeným blokem na uvedenou klávesu.

Pro přesun označeného bloku textu programu na jiné místo v programu budete muset použít schránku Windows. Pokud stisknete klávesy Ctrl+C zkopíruje se text z označeného bloku do schránky a blok zůstane v textu stále zachován. Naopak, pokud stisknete klávesy Ctrl+X, bude blok textu převeden do schránky Windows (z textu programu bude zrušen). Pokud se přesunete na požadované místo v programu, můžete do něj text ze schránky Windows zkopírovat. To se provede stiskem kláves Ctrl+V. Text ve schránce zůstává přitom stále zachován do doby dalšího načtení bloku. Pozor však na to, že obsah schránky Windows se může v ostatních aplikací Windows zrušit. Pro přesun textu mezi editorem a Windows jsou v menu systému zařazeny příslušné volby v menu Editace.

Stejně jako máte možnost přesouvat text mezi schránkou Windows a programem, můžete podobným způsobem převádět text povelů zadaných v příkazovém okně systému. Po dobu editace programu máte stále možnost používat příkazové okno a tím i například zkoušet parametry povelu. Až získáte správný výsledek, převedete odzkoušený povel přes schránku Windows do programu. Teoreticky můžete pomocí bloku převádět do programu v případě potřeby i obsah výstupního textového okna.

Až budete psát rozsáhlejší programy, budete občas potřebovat nalézt v textu programu určitý text. Po stisku kláves Ctrl+F, nebo položkou z menu Editace/Hledat se vám zobrazí dialogový box pro zadání parametrů hledání. Zadáte požadovaný text pro hledání, požadavek zda hledat pouze celá slova, jestli rozlišovat velká a malá písmena a směr hledání.

Podobným způsobem máte možnost zadat náhradu textu jiným textem. Dialogový box se zobrazí po stisku kláves Ctrl+L nebo po zadání volby menu Editace/Změnit. Nahrazovat přitom můžete pouze požadované výskyty hledaného textu, nebo komplet.

Pokud budete chtít program vytisknout, stiskněte klávesy Ctrl+P, nebo použijte volbu menu Editace/Tisk.

10.lekce - proměnné a jejich typy

Když jsme si ukazovali, jak vypsat příkazem WRITELN číslo do výstupního textového okna, bylo uvedeno, že není možné jednoduchým způsobem uložit výsledek výpočtu do paměti. Je to proto, že to nelze z příkazového řádku provést. Musí se použít program, ve kterém bude hodnota definována. V této lekci si proto ukážeme, jaké typy hodnot můžeme do paměti ukládat.

Hodnoty uložené v paměti se nazývají proměnnéHLP_0205. To proto, že jejich hodnoty se mohou programem měnit. Každou proměnnou musíme před jejím prvním použitím nejprve pojmenovat a definovat typ uložené hodnoty. Velmi zjednodušeně by se dalo napsat, že typ proměnné je buď číslo, znakový řetězec nebo logická hodnota. Pokud systém ví, s jakým typem proměnné má pracovat, je schopen provádět průběžné kontroly hodnot a hlavně zná, jaké místo má určité proměnné v paměti rezervovat.

Co je číslo, ví určitě každý. V systému je však definováno několik typů číselných proměnných, které se liší velikostí možných hodnot zpracovávaných čísel. Některé typy číselných proměnných jsou určeny pouze pro celá čísla, jiné mohou obsahovat i desetinnou část. Rozlišuje se také, že čísla mohou být buď pouze kladná, nebo i záporná. Požadovaný typ číselné proměnné je proto vždy nutno pečlivě zvážit. Znakový řetězec představuje seskupení libovolných znaků - písmen. Logická hodnota uchovává výsledky dotazu a může nabývat pouze hodnot pravda nebo nepravda.

Ještě dříve, než bude proměnná v programu poprvé použita, musí se v programu deklarovat. To znamená, že musíme uvést její jméno, typ a případně i počáteční hodnotu. Typ proměnné popisuje přitom množinu hodnot, které může nabývat a operace, které se s ní mohou provádět.

Základní proměnné jsou nejjednodušší proměnné, které reprezentují jeden výskyt prvku údaje. Ze základních typů proměnných jsou odvozovány další deklarace proměnných typů pole, záznam a tabulky, se kterými se seznámíme až později.

```
Podporované typy proměnných:
BYTE     - celočíselný typ s rozsahem 0 až 255
INTEGER  - celočíselný typ s rozsahem od -32768 do +32767
WORD     - celočíselný typ s rozsahem od 0 do 65535
LONGINT  - celé číslo  od -2  miliard do + 2 miliard
REAL     - číslo v rozsahu od 2.9*10-39 do 1.7*1038
STRING   - řetězec znaků o délce 255 znaků
CHAR     - jeden znak
BOOLEAN  - logická proměnná typu  BOOLEAN (True/False)
TEXT     - textový soubor typu Pascal TEXT
ARRAY    - jednorozměrné pole proměnných
```

Typ BYTE zabírá v paměti jednu slabiku a může proto obsahovat pouze celá čísla pro hodnoty od 0 do 255.

Datový typ INTEGER je v paměti uložen ve dvou slabikách a může proto nabývat hodnot od -32768 do +32767.

Typ WORD zabírá v paměti také dvě slabiky, ale protože může obsahovat pouze kladná čísla, může obsahovat čísla od 0 do 65535.

Typ LONGINT může nabývat záporných hodnot a protože obsahuje v paměti čtyři slabiky, může nabývat hodnot od mínus dvou miliard do plus dvou miliard.

Datový typ REAL se používá pro vyjádření hodnot s pohyblivou řádovou čárkou. Číslo je uloženo v šesti slabikách a může nabývat hodnot od 2.9*10-39 do 1.7*1038

Datový typ znak CHAR má velikost jedné slabiky a uchovává jeden znak. Znakové konstanty se vyjadřují tak, že se znak uzavře do apostrofů, například 'A', '1', '&'. Zápis '1' v tomto případě znamená znak pro vyjádření číslice jedna nikoli číselnou hodnotu. Na rozdíl od klasického jazyka PASCAL je možné znaky uzavírat i do uvozovek.

Datový typ STRING představuje posloupnost znaků s pevnou délkou 255 znaků typu CHAR bezprostředně za sebou. Vytváří tím tzv. řetězec. Řetězec musí být ohraničen apostrofy nebo dvojitou uvozovkou. Řetězec může obsahovat maximálně 255 znaků.

Datový typ BOOLEAN má pouze dvě možné hodnoty a to True (pro stav pravda) a False (pro stav nepravda). Lze mu proto v programu přiřadit buď konkrétní hodnotu True nebo False, případně výsledek vyhodnocení logického výrazu. Logické výrazy najdou uplatnění zejména v podmíněných příkazech, příkazech cyklu apod. a probereme si je v některé z dalších lekcích.

Dále jsou definovány ještě proměnné typu TEXT a ARRAY. Jimi se budeme podrobněji zabývat až později. V následující lekci si probereme, jak se proměnné v programu deklarují a jak s mimi pracovat.

11.lekce - deklarace a používání proměnných

V předchozí lekci jsme se seznámili s tím, že průběžné hodnoty výpočtů se ukládají proměnných. Popsali jsme si také, jaké typy proměnných mohou existovat. V této lekci si ukážeme, jak proměnnou v programu deklarovat a jak ji používat.

Proměnné musí být v programu deklarovány ještě před svým prvním použitím ve výpočtech. V deklaraci proměnné se definuje, název proměnné a její typ s možností uvedení počáteční hodnoty. Název proměnné musí začínat písmenem a nesmí obsahovat mezeru. V programu se musí před deklarací proměnných uvést klíčové slovo VAR a na konci ENDVAR. V tomto bloku deklarace proměnných není možné používat žádné povely. Vyzkoušejte si následující příklad:

```
VAR
   a : integer = 100;
   b : integer = 3;
   c : integer;
   d : real;
   s : string;
ENDVAR

PROCEDURE main
   c := a/b;                {výpočet   hodnoty}
   d := a/b;                {výpočet   hodnoty}
   s := 'OZOGAN KLONDAIK';  {přiřazení hodnoty}
   WRITELN(c:10:3);         {vypíše výsledek 3.000 }
   WRITELN(d:10:3);         {vypíše výsledek 3.333 }
   WRITELN(s);
ENDPROC
```

Proměnné a, b a c jsou deklarovány jako typ integer, to znamená, že mohou obsahovat pouze celá čísla. Proměnná d je typu real a může proto obsahovat reálné číslo včetně desetinných míst. Proměnná s je řetězec. V programu jsme proměnné nejprve v bloku označeném slovy VAR a ENDVAR deklarovali. U proměnné a, b jsme definovali současně počáteční hodnoty. V dalším bloku programu označeném bloky PROCEDURE a ENDPROC jsme deklarované proměnné použili.

Hodnotu proměnné stanovíme jejím přiřazením. To se provede pomocí dvojice znaků := mezi nimiž nesmí být mezera. Na levé straně je přitom uvedena proměnná, jejíž hodnota má být změněna a na straně pravé je uvedena přiřazovaná hodnota. Hodnotu můžeme uvést buď přímo (číslo, řetězec) nebo jako výraz. Co je to výraz si přesněji probereme v dalších lekcích. Nyní bude stačit, pokud si budete pamatovat, že výraz může být například matematický výpočet.

Po spuštění výše uvedeného programu si systém nejprve z části VAR..ENDVAR definuje použité proměnné a jejich typ a v druhé části programu PROCEDURE..ENDPROC s nimi provádí zadané příkazy. Nejprve do proměnné c uloží dělení a/b. Stejně tak učiní s proměnnou d. Po výpisu proměnných do výstupního okna si můžete myslet, že výpočet neproběhl správně. Obsah proměnných se liší, u proměnné c nelze vypsat desetinnou část výsledku. Pokud si však zkontrolujete typ proměnné c, zjistíte, že je vše v pořádku. Proměnná je deklarována jako integer, což znamená pouze celé číslo (kladné nebo záporné). Desetinná část výsledku se proto neukládá a není možné ji také vypsat.

Po ukončení programu zůstanou deklarované proměnné stále aktivní. Můžeme je proto i po ukončení programu dále používat a vypsat například příkazem WRITELN jejich obsah. Nyní máte proto možnost vyzkoušet si používání nadeklarovaných proměnných z příkazového okna. Novým spuštěním programu dojde k jejich deaktivaci (zrušení) a nahrazení novými proměnnými. Prozkoušejte si

využití i jiných typů proměnných deklarovaných v programu.

Dosud jsme sestavovali naše programy tak, že vykonávaly pouze zadanou posloupnost akcí bez možnosti vynechání některých akcí. V následující lekci se proto seznámíme s možností rozdělení činnosti programu na základě vyhodnocení zadané podmínky.

12.lekce - podmínky v programu, logické výrazy

Jak jsme již uvedli v úvodu probíraných lekcí, je programování vlastně sestavování posloupnosti vykonávaných akcí. V některých případech musíme mít ale možnost některou z akcí buď vynechat, nebo provést akce jiné. Obojí na základě zadané podmínky. Například pokud mám v kapse více než dvacet korun, půjdu do kina, jinak půjdu domů na televizi. Zadaná podmínka by se dala v lidské mluvě popsat následujícím vztahem:

```
POKUD hotovost > 20 POTOM
   půjdu do kina
JINAK
   budu se dívat na televizi
KONEC
```

Uvedenému zápisu se říká algoritmus a popisuje schematicky postup prováděných akcí. Vztahu hotovost>20 se říká podmínka. Podmínka nám tedy určuje, jaká akce se bude provádět. Zápis algoritmu je nutné pro počítač převést do programu. Například následujícím způsobem:

```
VAR
   hotovost : integer;
ENDVAR

PROCEDURA main
   hotovost := 15;          {zadejte vaši hotovost}
   IF hotovost > 20 THEN
      WRITELN("Kino")       {pokud je podmínka splněna}
   ELSE
      WRITELN("Televize")   {pokud není podmínka splněna}
   ENDIF;
ENDROC
```

Nejprve jsme deklarovali proměnnou se jménem hotovost. Dále jsme jí v programu přiřadili určitou hodnotu. V podmínce jsme potom zjišťovali, zda odpovídá našim požadavkům a podle toho jsme provedli určitou akci. Zkuste si zadat sami různé hodnoty stavu vaší hotovosti. Pozor ale na to, že hotovost je deklarována jako typ integer. Nesmíte být proto moc bohatí a mít v hotovosti více než 32767 Kč. Zkuste upravit program tak, aby jste mohli zadávat i větší částky. Závisí to na typu proměnné.

Všimněte si, že jsme v programu u zápisu podmínky odsadili vykonávané příkazy na řádku o tři znaky. Tím vynikla struktura podmínky a na první pohled je viditelné její rozčlenění. Není to sice nutnost, přesto však doporučujeme uvedenou grafickou podobu zápisu programu ve vlastním zájmu dodržovat.

Podmínka se v programu zadává klíčovým slovem IF, za kterým musí následovat vyhodnocovaná podmínka. Klíčové slovo THEN není povinné, přesto jej však doporučujeme uvádět, protože to vyžaduje klasický jazyk Pascal. Dále následují povely, které se provedou pouze při splnění podmínky. Za klíčovým slovem ELSE se uvedou povely, které se provedou při nesplnění podmínky. Zápis podmínky je ukončen klíčovým slovem ENDIF. V zápise podmínky můžete vynechat klíčové slovo ELSE včetně akcí pro nesplnění podmínky. Vždy ale musíte uvést ukončení podmínky klíčovým slovem ENDIF. Pokud se tak nestane, nahlásí systém chybu v programu.

Podmínku představuje logický výraz, který udává, zda je podmínka splněna nebo ne. Logické výrazy mohou mít proto výsledek pouze pravda nebo nepravda. V počítačové terminologii True (pravda) nebo False (nepravda). Pokud by jste chtěli výsledek podmínky deklarovat jako proměnnou,

museli by jste použít typ BOOLEAN. Logická podmínka zpracovává nejčastěji matematický výraz. Může to však být i výraz zpracovávající řetězce, to však bude psáno až se naučíme s rětězci důkladněji pracovat.

Matematické výrazy porovnávají nejčastěji několik hodnot. Ve výrazu se přitom může také použít libovolného matematického výpočtu, který je v jazyce definován.

Matematické výrazy zpracovávají aritmetické operace. Výrazy se skládají z operátorů a operandů. Operátor je přitom porovnávaná hodnota a operand je způsob porovnání hodnot. Operátor může představovat libovolný matematický výpočet, který je v jazyce definován. Operand slouží k vyhodnocení operátorů. Při zápisu dodržte, aby byly operátory odděleny od operandů mezerami !

Používají se následující dostatečně známé operandy:

```
>        větší než
>=       větší nebo rovno než
<        menší než
<=       menší nebo rovno než
=        rovno
<>       nerovno
```

Ve výrazu je samozřejmě možné používat závorky. Příklad výrazů zpracovávajících matematický výpočet:

```
hotovost >   20
  a/2    <   10
2*(a+b)  = 2*a+2*b
```

Výrazy můžeme dále v jedné podmínce spojit logickým operátorem s dalším výrazem a vyhodnocovat tak složenou podmínku. Používají se přitom následující logické operátory:

```
AND         a zároveň platí
OR          platí jeden nebo druhý výraz
NOT         není pravda, negace výrazu
```

V případě použití logických operátorů mají tyto ve vyhodnocování výrazů přednost před ostatními operátory. Dále se vyhodnocují závorky a až na konci matematické výpočty. Pokud budete chtít zapsat v programu několik výrazů spojených logickým operátorem, musíte umístit výrazy do závorek. Například pro zjištění rozsahu hotovosti od 10 Kč do 30 Kč použijete následující zápis:

```
IF (hotovost >= 10) and (hotovost <= 30) THEN
   WRITELN("Kino")        {pokud je podmínka splněna}
ELSE
   WRITELN("Televize")    {pokud není podmínka splněna}
ENDIF;
```

V této lekci jsme si probrali mimo možnosti rozvětvení činnosti programu také způsob zápisu podmínek v programu. Podmínky se v programu používají i v dalších příkazech. Například pro zadání počtu opakování zvolené části programu, jak si ukážeme v následující lekci.

13.lekce - programové cykly

Až doposud prováděly naše programy pouze zadaný sled povelů bez možnosti opakování požadovaných akcí. Pokud by jsme potřebovali například vykonat nějakou část programu stokrát, museli by jsme uvedenou část programu zapsat stokrát. To by však bylo, jak sami jistě uznáte značně nevhodné. V této lekci si proto probereme, jak je možné na základě vyhodnocení podmínek opakovat zadanou část programu.

Pokud budete potřebovat v programu provést nějakou akci s předem známým počtem opakování, bude nejvhodnější použít cyklus typu FOR..ENDFOR. Jedná se o cyklus, ve kterém je definován tzv. čítač, který udává kolikrát se má cyklus ještě provést. Čítač je automaticky systémem zvyšován o zadanou hodnotu. Směr načítání může být přitom nahoru, nebo dolů:

```
VAR
   x : real
ENDVAR

PROCEDURE main
   ConsoleClear;
   FOR x := 1 to 3 step 0.5  {cyklus nahoru, udán krok}
      WRITELN(x:10:1);
   ENDFOR

   FOR x := 5 downto 1       {cyklus směrem dolů}
      WRITELN(x:10);
   ENDFOR
ENDPROC
```

V uvedeném příkladě jsou v programu dva cykly FOR. První z nich zvyšuje stav čítače x směrem nahoru, druhý směrem dolů. Všimněte si, že čítač musí být sice deklarován jako proměnná, ale přiřazení počáteční hodnoty se provádí až v definici cyklu. Konečná hodnota čítače se pro směr načítání nahoru zadává za klíčovým slovem 'to'. Pro směr načítání dolů je určeno klíčové slovo 'downto'. Pokud chceme zvyšovat, nebo snižovat stav čítače o jinou hodnotu než jedna, musíme požadovanou hodnotu uvést za klíčovým slovem 'step'. Hodnota řídící proměnné se nesmí v cyklu měnit. Hodnoty cyklu jsou vyhodnocovány pouze jednou a to při spuštění příkazu FOR.

Důležitá poznámka: proměnná x je v předchozím příkladě deklarována jako typ real, to znamená že může nabývat i desetinných hodnot. Pokud by jste uvedli v tomto příkladě proměnnou x jako typ integer, zvyšoval by se čítač vždy sice o hodnotu 0.5 ale protože typ integer je pouze celočíselný, ke zvýšení čítače by nikdy nedošlo a program by skončil v nekonečné smyčce. Museli by jste jej restartovat pomocí volbou z menu Program/Restart programu.

Pokud budete chtít ve svém programu použít cyklus, jehož ukončení bude záviset na splnění zadané podmínky, máte možnost použít cyklus typu REPEAT..UNTIL. Vnitřní část tohoto cyklu se provede vždy minimálně jednou, protože podmínka ukončení cyklu je uvedena až na jeho konci:

```
VAR
   x : real
ENDVAR

PROCEDURE main
   ConsoleClear;
   x := 1;
```

```
   REPEAT                {začátek cyklu}
      WRITELN(x/(x+1):10:3);
      x := x+1;          {zvýšení hodnoty čítače}
   UNTIL x = 5           {dokud není výraz pravdivý}
ENDPROC
```

Příkazy uvedené mezi klíčovými slovy REPEAT a UNTIL se provádí tak dlouho, dokud není výraz definovaný na konci podmínky pravdivý. Nesmíte přitom zapomenout na zvýšení hodnoty čítače, případně provést jinou akci, která může mít za následek ukončení cyklu.

Jako další cyklus můžete použít WHILE..ENDWHILE, který vyhodnocuje podmínku opakování cyklu na jeho počátku. To umožňuje, že pokud není podmínka splněna, neprovedou se příkazy uvedené mezi klíčovými slovy WHILE a ENDWHILE ani jednou. To může být v některých případech výhodné a je proto nutné se při vlastním programování rozhodnout, který druh cyklu má být použit.

```
VAR
   x : integer
ENDVAR

PROCEDURE main
   ConsoleClear;
   x := 1;
   WHILE x < 5           {dokud je výraz pravdivý}
      WRITELN(x*3:10);
      x := x+1;          {zvýšení hodnoty čítače}
   ENDWHILE              {konec cyklu}
ENDPROC
```

Pro zápis podmínek při používání cyklů platí vše co bylo uvedeno u popisu větvení programu příkazem IF..ENDIF. Stejně tak je velmi výhodné a hlavně přehledné v textu programu odsadit řádky programu uvnitř cyklů o tři prázdné znaky doprava. Viz výše uvedené příklady. Programy tím získají na přehlednosti. Současně doporučujeme uvádět v programu poznámky ve formě komentáře. Jak jste si již mohli všimnout, poznámky se uvádějí ve složených závorkách. Pokud by jste na začátku řádku uvedli dvě lomítka, byl by text až do konce řádku v programu ignorován.

V některých případech můžete potřebovat, aby se cyklus WHILE, REPEAT nebo FOR předčasně přerušil a pokračoval dalším cyklem od začátku. K tomu se používá příkaz CONTINUE, který je vyhodnocen jako ENDWHILE, UNTIL nebo ENDFOR a řízení programu je předáno zpět na začátek cyklu. Klíčové slovo je přitom uvedeno až za příkazy, které se mají provést vždy, ještě před ukončením (přerušením) cyklu. Pokud se příkaz použije mimo smyčku, je vyvoláno chybové hlášení. Následující příklad kreslí pomocí vykreslování bodů čáry s přerušením uprostřed.

```
VAR
   x : integer
ENDVAR

PROCEDURE main
   ImageInit(100,100);    {Inicializuje grafickou plochu}
   ImageShow;             {zobrazí grafický výstup }
   ImagePenColor(clBlue); {nastaví modrou barvu}
   FOR x := 1 to 100;
       Point(25,x,3);
```

```
        Point(75,x,3);
        IF (x > 25) and (x < 75) then
           CONTINUE;      {přeruší cyklus s návratem}
        ENDIF
        Point(50,x,3);    {pro x od 25 do 75 se neprovede !}
    ENDFOR
    ImagePenColor(clRed);          {nastaví červenou barvu}
    Point(GetmaxX/2,GetmaxY/2,30); {nakreslí bod doprostřed}
ENDPROC
```

V této lekci jsme se zabývali několika příkazy, které však prováděly podobnou činnost. Příkazy cyklu jsou v programech velmi často používané. Je však nutné si podle požadovaných akcí vybrat nevhodnější příkaz pro cyklus.

14.lekce - zadávání vstupních hodnot

U programů v předchozích lekcích jsme vždy počítali s tím, že jsou všechna data pro výpočty zadána přímo v programu. Aby jsme ale měli možnost ovlivnit průběh výpočtu, bylo by vhodné zadávat požadované hodnoty přímo dotazem od uživatele. To je možné dvěma způsoby. Buď použijeme příkaz READ, který umožňuje mimo jiné uživatelské zadaní hodnoty proměnné, nebo složitější postup, kdy použijeme interaktivní knihovnu pro tvorbu formulářů.

Příkaz READ se používá pro vstup hodnoty proměnné ze souboru, nebo od uživatele z klávesnice. Hodnotu potom můžeme použít dále v programu k výpočtům. Proměnná musí být nejprve definována. Možné je použít pouze číselné a řetězcové proměnné. Čtení hodnoty ze souboru si ukážeme později, nyní si předvedeme, jak zadat proměnnou z klávesnice.

Příkaz READ vyvolá jednoduchý formulář s možností zadání hodnoty proměnné. Jméno formuláře je 'vstup' a pokud by jste chtěli umístit nad editační box pro zadání hodnoty proměnné váš text, museli by jste text vypsat příkazem WRITE (pozor, ne WRITELN !) ještě před použitím příkazu READ. Zadaný text se samozřejmě také vypíše do textového výstupního okna. Toho lze využít pro následné zadání výstupních hodnot. Pokud by vám výpis textu vadil, je možné nejprve navstupovat požadované vstupní hodnoty, vymazat obsah výstupního okna a až potom provést výpočty a zobrazení výsledků. Následuje příklad použití příkazu READ.

```
VAR
   s : string;
   r : real;
ENDVAR

PROCEDURE main
   ConsoleClear;
   WRITE("Zadejte celé číslo");
   READ(s);
   WRITELN();
   WRITELN("Zadal jste číslo:",StrToInt(s):12);
   WRITELN();

   WRITE("Zadejte libovolné číslo");
   READ(r);
   WRITELN();
   WRITELN("Zadal jste číslo:",r:12:4);
   WRITELN();

   WRITE("Zadejte libovolný řetězec");
   READ(s);
   WRITELN();
   WRITELN("Zadal jste řetězec: ",s);
   WRITELN();
ENDPROC
```

Se složitějším, ale efektnějším způsobem se seznámíme při probírání interaktivní knihovny, která umožňuje definovat v programu vzhled i obsah formuláře. Zkuste si proto zatím alespoň následující příklad:

```
VAR
```

```
   jmeno : string = '';
   rok1  : integer = 1900;  rok2  : integer;
   mesic : integer;
   den1  : integer; den2  : integer;
ENDVAR;

ROCEDURE formular;
   createForm('form1',"Vítám Vás !", 100, 100, 270, 180);
   addStatic ('form1',"Dobrý den,", 10, 15, 200, 14);
   addStatic ('form1',"jaké je Vaše jméno ?",10,45,200,14);
   addEditBox('form1','jmeno', 'jmeno', 10, 80, 250, 20);
   addStatic('form1',"narodil jste se v roce:",10,120,200,14);
   addEditBox('form1','rok1', 'rok1', 210, 120, 40, 20);
ENDPROC;

PROCEDURE main
   formular;
   IF (formDialog("form1"))
      ConsoleClear;
      WRITELN('Dobrý den ' + jmeno, ', přeji pěkný den !');
      GetDate(rok2, mesic, den1, den2);
      WRITE("dnes je ", den1, "/", mesic, "/", rok2, ", ");
      WRITELN("je Vám ", rok2-rok1, " roků.");
   ENDIF
ENDPROC
```

Takže jak zadávat obsah proměnné od uživatele již známe. Víme také, jak vypsat jejich obsah s případnou další úpravou do textového výstupního okna. Zatím jsme se ale nic nedozvěděli o tom, jak je možné provést výstup informací na tiskárnu. Systém sice neobsahuje žádné příkazy pro přímý tisk na tiskárnu, dovoluje vám ale vytisknout kompletní obsah textového výstupního okna. Musíte proto nejprve provést výstup požadovaných informací a textů do výstupního okna a potom příkazem ConsolePrint vytisknout jeho obsah. Můžete také po ukončení programu, kdy se zobrazí požadované výsledky ve výstupním okně provést jeho tisk pomocí ikony s obrázkem tiskárny, umístěné v horní liště textového výstupního okna.

15. lekce - vícenásobné větvení programu

V jedné z předchozích lekcí jsme se seznámili s příkazem IF..ENDIF, který slouží pro větvení činnosti programu. Poznali jsme, že můžeme činnost programu rozdělit do dvou větví, podle pravdivosti zadané podmínky. Pokud by jste však potřebovali rozdělit činnost programu podle výsledku na více větví, byl by zápis pomocí příkazu IF velmi nepřehledný. Proto systém obsahuje příkaz SWITCH, který můžete použít pro rozdělení činnosti programu do libovolného počtu větví na základě vyhodnocení výsledku zadané podmínky. Pokud budete například potřebovat pro každé číslo samostatnou akci, můžete tak učinit podle následujícího příkladu:

```
VAR
   x : integer
ENDVAR

PROCEDURE main
   ConsoleClear;
   FOR x := 1 to 10;
       SWITCH x OF              {porovnávej proměnnou x }
          CASE 1:               {pokud má hodnotu 1 }
               WRITELN("číslo jedna");
          ENDCASE
          CASE 5:               {pokud má hodnotu 5 }
               WRITELN("číslo pět");
          ENDCASE
          ELSE                  {pokud má jinou hodnotu}
               WRITELN(x:10);
          ENDCASE
       ENDSWITCH
   ENDFOR
ENDPROC
```

Za klíčové slovo SWITCH se v programu uvádí hodnota, nebo výraz, který se potom následně povelem CASE vyhodnocuje. Pokud se hodnota, nebo výsledek výrazu rovná některému z výrazů uvedených za klíčovým slovem CASE, bude proveden blok příkazů za tímto výrazem. Program bude potom pokračovat po ukončení konstrukce příkazu klíčovým slovem ENDSWITCH. Pokud se výraz nebude rovnat žádnému z porovnávaných výrazů, provede se blok příkazů uvedený mezi klíčovými slovy ELSE a ENDCASE.

Jedná se o poměrně dosti složitou konstrukci jazyka. Pro názorné předvedení proto můžete program spustit klávesou F6 v animačním režimu, kdy systém zobrazuje postupně řádek programu, který právě vykonává. Všimněte si přitom, že řádky, které nevyhovují zadání jsou přeskakovány. Pokud nebudete stačit sledovat animaci, můžete zkusit krokování programu po jednotlivých řádcích. Po stisku klávesy F7 se vykoná vždy pouze jeden řádek programu.

16.lekce - struktura programu

Až dosud se naše programy skládaly pouze ze dvou částí. V první části uvedené mezi slovy VAR a ENDVAR jsme deklarovali dále používané proměnné. Ve druhé části se mezi klíčovými slovy PROCEDURE a ENDPROC uváděly jednotlivé příkazy programu. Takovým částem programu se říká bloky. Začátek i konec bloku je vždy definován příslušnými klíčovými slovy pro začátek a konec bloku. V programu mohou být mimo deklarací a hlavního příkazového bloku uvedeny i další části, se kterými jsme se dosud neseznámili. Viz následující schématická struktura programu:

```
TYPE
   {definice datových typů - záznamů}
ENDTYPE

VAR
   {deklarace proměnných}
ENDVAR

PROCEDURE ...
   {deklarace uživatelské procedury}
ENDPROC

FUNCTION ...
   {deklarace uživatelské funkce}
ENDPROC

PROCEDURE main
   {tělo hlavního programu}
ENDPROC
```

Definice datových typů

Datové typy jsou uživatelsky definované typy proměnných, které umožňují například seskupovat několik proměnných a používat je jako jednu strukturovanou proměnnou. Podrobněji budou popsány později.

Deklarace proměnných

Již znáte. Jsou uvedeny v bloku mezi klíčovými slovy VAR a ENDVAR. Obsahují deklarace proměnných, určení jejich typu a případné stanovení počáteční hodnoty. Podrobněji se k nim ještě vrátíme později.

Deklarace uživatelských procedur

Procedury jsou uživatelsky definované příkazy, které umožní lépe rozčlenit program do logických celků a vícenásobné použití části kódu. Podrobněji se budeme procedurami zabývat v následující lekci.

Deklarace uživatelských funkcí

Funkce se používají pro uživatelskou definici zpracování výrazů s možností vícenásobného použití v programu. Podrobněji se budeme funkcemi zabývat v následující kapitole.

Tělo hlavního programu

Musí být uvedeno v každém programu a musí být obsaženo v proceduře se jménem main, kterou doporučujeme umístit vždy na konec programu. Jejím prováděním se začíná vždy činnost programu. Pokud by nebyla procedura se jménem main v programu nalezena, systém by nahlásil chybu a program by byl ukončen.

Standardní jazyk Pascal dodržuje mnohem přísnější formát zápisu programu. Platí přitom obecné pravidlo, že každý prvek programu se musí v jazyce Pascal nejprve deklarovat a až potom se může používat. Bloky v programech v jazyce Pascal jsou vyznačeny klíčovými slovy Begin a End místo námi používaného ukončení ENDIF, ENDFOR, ENDPROC a podobně. Námi používané řešení vychází z jazyků používaných pro programování databází. Má tu výhodu, že je mnohem přehlednější a rychleji se jej naučíte používat.

Bloková struktura programu se dodržuje i u dalších, dříve probraných příkazů. Jedná se o rozhodovací příkaz IF..ENDIF a všechny příkazy cyklů FOR..ENDFOR, REPEAT..UNTIL a WHILE..ENDWHILE. Všechny struktury programu musí být ukončeny svým příslušným ukončením. Bloky mohou být do sebe vnořovány, nesmí však docházet k přesahům jejich konců. Proto je vhodné dodržovat grafickou úpravu programu, kdy jsou podřízené části bloků odsazeny od svého počátku a konce. Tím se dosáhne současně přehlednosti programu.

Jak již bylo uvedeno, musí být v programy vždy definována procedura se jménem main. Co jsou to procedury a jak je můžeme využít si ukážeme v následující lekci.

17.lekce - deklarace a používání procedur

Doposud jsme v našich programech používali pouze příkazy zabudované v systému. Nyní si ale předvedeme, jak si můžeme naprogramovat vlastní příkazy. Určitě jste se již setkali s tím, že pokud by jste seskupili několik řádků programu, které se v uvedené sestavě v programu několikrát opakují do jednoho, byl by program přehlednější a kratší. Na to můžete použít procedury, kterým se také někdy říká podprogramy.

Procedury jsou volány z hlavního programu, vykonají příkazy zadané ve svém těle procedury a po jejím ukončení předají vykonávání programu zpět do hlavního programu. Procedury jsou v programu označeny klíčovými slovy PROCEDURE a ukončeny klíčovým slovem ENDPROC, mezi nimiž jsou uvedeny výkonné příkazy procedury. Každá procedura musí být pojmenována. V programu se přitom nesmí vyskytovat více procedur se stejným jménem. Následuje příklad jednoduché procedury:

```
VAR
   a : integer = 10;
   b : integer =  3;
ENDVAR

PROCEDURE nadpis
   WRITELN("**************************");
   WRITELN("* počítá systém KLONDAIK * ");
   WRITELN("**************************");
ENDPROC;

PROCEDURE main
   nadpis;
   WRITELN("součet :", a+b:10:3):
   nadpis;
   WRITELN("rozdíl :", a-b:10:3):
   nadpis;
   WRITELN("násobek:", a*b:10:3):
ENDPROC
```

V příkladě jsme definovali proceduru se jménem nadpis, která vždy mezi výpočty vypíše tři řádky nadpisu. V hlavní proceduře main již potom stačí zadat volání procedury nadpis. Systém v tom okamžiku vypíše vždy tři řádky nadpisu. Tím jsme ušetřili místo a program je přehlednější.

Jistě budete souhlasit s tím, že možnost používání procedur je zajímavá vlastnost systému, že by ale byla ještě zajímavější, pokud by procedura dokázala pracovat s různými vstupními hodnotami. To je také možné, musíme však ve volání procedury i v její deklaraci uvést seznam parametrů a upravit příkazy uvnitř procedury tak, aby dokázaly s proměnnými parametry pracovat. Pokud by jste například potřebovali nakreslit do grafického výstupního okna na zadanou pozici čtverec s délkou hrany 20 bodů, mohli by jste použít následující příklad:

```
VAR
   k : integer;
ENDVAR

PROCEDURE ctverec(x : integer, y : integer);
   Rectangle(x, y, x+20, y+20);
```

```
ENDPROC

PROCEDURE main
   ImageInit(100,100);
   ImageBrushColor(clYelow);
   Rectangle(5, 5, 95, 95);
   ImageBrushColor(clRed);
   FOR k:= 10 to 70 step 30
       ctverec(10, k);
       ctverec(40, k);
       ctverec(70, k);
   ENDFOR
ENDPROC
```

Procedura ctverec je volána se dvěmi parametry uvedenými v závorce, oddělenými čárkou. Procedura má ve své definici deklarovány proměnné x a y, které se dále v proceduře používají. Do uvedených proměnných se převedou hodnoty zadané ve volání procedury. Procedura potom se zadanými hodnotami pracuje. Tak je umožněno předávat proceduře ke zpracování vstupní hodnoty.

Pokud má procedura pracovat s předávanými parametry, je nutné v deklaraci procedury uvést za jejím jménem v závorce seznam použitých proměnných. V seznamu se uvádí jméno proměnné a za dvojtečkou její typ. Proměnné jsou v seznamu oddělovány čárkami. Počet deklarovaných proměnných v proceduře musí přitom vždy odpovídat počtu předávaných parametrů ve volání procedury.

Zkuste nyní upravit proceduru ctverec i její volání tak, aby jste mohli zadávat ve volání procedury i délku strany čtverce. Musíte proto přidat do volání procedury další parametr. Stejně tak budete muset přidat parametr do deklarace procedury. Parametr potom použijete pro zadání délky hrany čtverce.

Pokud v programu nadeklarujete svou vlastní proceduru a program spustíte, zachová si systém informace o deklarované proceduře i po ukončení programu. Můžete potom proceduru použít z příkazového okna jako součást systému. To je velmi výhodná vlastnost, protože vám umožňuje seznámit se dokonale s možnostmi vlastnosti jazyka.

Pomocí deklarace procedur dosáhnete toho, že si můžete definovat své vlastní příkazy. Je přitom možné volané proceduře zadávat předávané parametry. Není ale možné, aby procedura vracela zpět výsledek provedení procedury. To vám umožní až funkce, které budou probrány v následující lekci.

18.lekce - deklarace a používání funkcí

V předchozí lekci jsme si ukázali, jak si nadefinovat vlastní příkaz ve formě procedury. Mnohdy by ale bylo výhodné, pokud by jsme mohli volat proceduru jako součást výrazu tak, aby nám vrátila procedura výsledek, se kterým by jsme ve výrazu dále pracovali. Něco takového je sice možné, používají se k tomu ale funkce.

Funkce je definovaná část programu, která je volaná z výrazu a provádí určité výpočty, případně akce. Při ukončení funkce nám vrací požadovaný výsledek, se kterým se ve výrazu, odkud byla funkce volána dále pracuje. Uveďme si proto příklad deklarace a použití funkce v programu, který nám provádí výpočet mocniny zadaným exponentem (53 = 5 x 5 x 5, výsledek je 125).

```
VAR
   k : integer;
   v : real;
ENDVAR

FUNCTION umocni(x : integer, y : integer): integer
   v := x;          {převezneme základ}
   FOR k := 1 to y
       v := v*x;    {násobíme opakovaně základem}
   ENDFOR
   umocni := v;     {předáme výsledku funkce}
ENDFUNC

PROCEDURE main
   ConsoleClear;
   WRITELN(umocni(2,9):10);  {vypíše hodnotu 1024}
   WRITELN(umocni(3,3):10);  {vypíše hodnotu   81}
   WRITELN(umocni(5,2):10);  {vypíše hodnotu  125}
ENDPROC
```

Funkce se od procedury liší svým zahájením a ukončením, kdy je výpočet funkce uveden mezi klíčová slova FUNCTION a ENDFUNC. V deklaraci funkce se navíc od procedury musí uvést za seznamem přebíraných parametrů i typ výsledku, který funkce vrací.

Další odlišností je způsob vytvoření hodnoty výsledku funkce. To je výsledku, který bude předán zpět. Jsou přitom dvě možnosti. První z nich je standardní a používá ji i jazyk Pascal a proto ji doporučujeme preferovat. Na konci zpracování funkce zapíšeme přiřazovací příkaz:

```
identifikátor := výraz;
```

kde na levé straně přiřazovacího příkazu je uvedeno jméno funkce a na pravé straně její výsledná hodnota. Pozor na to, že identifikátor přitom není deklarován jako proměnná. Je to proto, že stejně jako si systém deklaruje sám proměnné, které se předávají jako parametry procedury a funkce, stejně tak si systém deklaruje i proměnnou se jménem deklarované procedury.

Druhou možností pro předání výsledné hodnoty je příkaz RETURN s uvedením návratové hodnoty:

```
FUNCTION umocni(x : integer, y : integer): integer
   v := x;          {převezneme základ}
   FOR k := 1 to y
```

```
        v := v*x;     {násobíme opakovaně základem}
    ENDFOR
    RETURN v;         {předáme výsledek funkce}
ENDFUNC
```

Uvedené použití vychází z jazyka BASIC a databázových jazyků. Může být v některých případech jednodušší, jak však již bylo uvedeno, nejedná se o standardní postup jazyka Pascal.

Pokud v programu nadeklarujete svou vlastní funkci a program spustíte, zachová si systém informace o deklarované funkci i po ukončení programu. Můžete potom funkci použít u příkazů z příkazového okna jako součást systému. To je velmi výhodná vlastnost, protože vám umožňuje seznámit se dokonale s možnostmi vlastnosti jazyka.

19.lekce - lokální a globální proměnné

V předchozích lekcích jsme v programu deklarovali proměnné v samostatném bloku umístěném na začátku programu. Nadeklarované proměnné přitom platily po celou dobu programu a zůstávaly aktivní dokonce i po ukončení programu. Tomu se říká globální platnost proměnné, protože je dosažitelná kdykoliv z libovolné procedury v programu. Opakem je proměnná deklarovaná pouze s lokální dobou platnosti. Pro dokonalejší seznámení se s lokálními a globálními proměnnými si upravíme dříve uvedený příklad z lekce zabývající se funkcemi do následující podoby:

```
VAR
   v : integer = 999;
ENDVAR

FUNCTION umocni(x : integer, y : integer): integer
   VAR
      k : integer;
      v : real;
   ENDVAR
   v := x;           {převezmeme základ}
   WRITELN("předávaná hodnota x:",x:4);
   WRITELN("lokální   hodnota v:",v:4);
   FOR k:= 1 to y
       v := v*x;     {násobíme opakovaně základem}
   ENDFOR
   umocni := v;      {předáme výsledku funkce}
ENDPROC

PROCEDURE main
   ConsoleClear;
   WRITELN("globálnní hodnota v:",V:4);
   WRITELN(umocni(2,9):10);  {vypíše hodnotu 1024}
   WRITELN(umocni(3,3):10);  {vypíše hodnotu   81}
   WRITELN("globálnní hodnota v:",V:4);
ENDPROC
```

Všimněte si, že program obsahuje dva bloky s deklarovanými proměnnými. Deklarační blok na začátku programu obsahuje pouze proměnnou 'v' s přidělenou hodnotou. Tato proměnná, protože je deklarována mimo jakékoliv procedury je globální a dosažitelná v celém programu. V definované funkci umocni je uveden znovu blok definice proměnných, kde je opět deklarována proměnná se jménem 'v' (dokonce rozdílného typu). Tato proměnné je pouze lokální, to znamená, že je platná pouze ve funkci, kde byla deklarována. Pokud deklarujeme lokální proměnou stejného jména, jaké má již existující globální proměnná, zastíní lokální proměnná po dobu své platnosti dříve definovanou globální proměnnou.

Pokud výše uvedený program spustíme, bude se mimo výpočtu vypisovat i obsah zadaných proměnných. Všimněte si, že pokud vypíšeme stav proměnné 'v' z hlavní procedury main, bude vypsána vždy hodnota, která byla přidělena proměnné při její deklaraci. Vypsána je tedy hodnota proměnné s globální platností v celém programu. Pokud se bude vypisovat hodnota proměnné 'v' z funkce umocni, bude se vypisovat hodnota lokální proměnné, která dočasně překryje globální proměnnou. Můžeme také vypsat hodnotu proměnné 'x', kterou jsme sami ani nedeklarovali. Deklaroval si ji opět dočasně sám systém pro převedení hodnoty z nadřazeného programu.

Možná se vám bude zpočátku zdát používání a sledování lokálních a globálních proměnných nepřehledné. Jejich používání ale umožní zpřehlednění programu. V některých případech je správné pochopení rozsahu platnosti lokálních a globálních proměnných dokonce nutné. Je to například problém rekurze, kdy z těla funkce voláme stejnou funkci (volání funkce sebe sama).

20. lekce - rekurze v programu

Rekurzívní funkce (nebo procedura) během provádění příkazů svého těla vyvolá ještě před jeho ukončením sama sebe. Znovu se tedy začne provádět tatáž posloupnost příkazů, aniž by původní byla dokončena. Pokud budeme chtít například vypočítat faktoriál nějakého čísla, musíme postupně vynásobit požadované číslo všemi čísly nižšími vždy o jedničku. Faktoriál čísla 5 se například vypočítá vztahem 5*4*3*2*1. Pokud budeme chtít pro výpočet faktoriálu napsat obecnou funkci, můžeme vztah upravit do následujícího tvaru: 5*(4*(3*(2*(1)))). Zde je již vidět, že je možné při násobení snížit vždy základ o jedničku a násobit opakováním výsledku volané funkce. Funkce pro výpočet faktoriálu bude mít proto následující zápis:

```
FUNCTION factorial(n: real): real;
   IF (n = 0) then
      factorial := 1
   ELSE
      factorial := n*factorial(n-1);   { REKURZE ! }
   ENDIF
ENDFUNC;

PROCEDURE main
   ConsoleClear;
   WRITELN('Test rekurze:');
   WRITELN('faktoriál čísla 1 je:', factorial(1):5); {=   1}
   WRITELN('faktoriál čísla 2 je:', factorial(2):5); {=   2}
   WRITELN('faktoriál čísla 3 je:', factorial(3):5); {=   6}
   WRITELN('faktoriál čísla 4 je:', factorial(4):5); {=  24}
   WRITELN('faktoriál čísla 5 je:', factorial(5):5); {= 120}
   WRITELN('faktoriál čísla 6 je:', factorial(6):5); {= 720}
   WRITELN('faktoriál čísla 7 je:', factorial(7):5); {=5040}
ENDPROC;
```

Všimněte si, že uvedený program neobsahuje žádnou deklaraci proměnných a přesto je používá. Je to proto, že proměnné jsou deklarovány systémem v deklaraci funkce.

Má-li rekurzivně aktivovaná funkce lokální proměnné, je při každé aktivaci vytvořena nová sada lokálních proměnných, přičemž při nové aktivaci zůstanou lokální proměnné nadřazených aktivací zachovány. Každá aktivace funkce má tedy své lokální proměnné. K proměnným nadřazených aktivací téže procedury či funkce není přístup možný. Po ukončení aktivace zaniknou odpovídající lokální proměnné a platnými se stanou lokální proměnné nadřazené aktivace.

Existují úkoly, jejichž řešení vede na zápis programů s rekurzivními procedurami nebo funkcemi. Jsou to problémy, při jejichž rozkladu na podproblémy vznikne problém, který je obdobný původnímu, avšak je jednodušší. S tím také souvisí následující lekce, kde se budeme věnovat návrhu programu a rozkladu zadání na jednotlivé procedury.

21. lekce - návrh a zápis programu

V předchozích lekcích jsme se seznámili se základními příkazy systému a strukturou programu. Poznali jsme, že programHLP_0201 tvoří posloupnost operací, větvení programu podle podmínek a opakování částí programu. Poznali jsme, že program tvoří i data, která program zpracovává. Nyní by bylo vhodné seznámit se s tím, jak by měl nový program vznikat.

Ještě dříve, než začneme psát nový program by jsme si měli ujasnit požadovanou činnost programu. Nejlépe tak, že se seznámíme nejprve s tím, jaké výsledky nám má program poskytnout. Podle toho si musíme zadat počáteční podmínky a vstupní hodnoty. Dalším krokem by již měl být rozklad problému tak, aby jsme si v blocích definovali činnost programu od počátečních hodnot až k požadovaným výstupním hodnotám. Pokud například zjistíme, že musíme nejprve provést výpočet vstupních hodnot a potom výsledné hodnoty zobrazit, můžeme již začít psát kostru programu:

```
PROCEDURE main
   vypocet;
   zobrazeni;
ENDPROC
```

Tím máme současně zadány jména procedur, které musíme naprogramovat. Dále již budeme provádět postupně rozklad dílčích problémů na další podproblémy. V případě složitých zadání se může dokonce stát, že nejsme schopni sestavit ihned program pro vyřešení dílčích problémů, ale musíme znovu definovat rozklad problému na menší související posloupné akce. Uvedeným způsobem se snažíme rozložit počáteční zadání problému do postupných kroků, které dále zjemňujeme. Na konci by jsme se měli dostat do fáze, že řešení již zapisujeme přímo v programovém jazyce. Souběžně s tím upřesňujeme požadovaná data a jejich uložení v proměnných definovaného typu.

Výše uvedenému způsobu řešení programů se říká metoda návrhu programu shora dolů. To proto, že od celkového zadání přecházíme postupně k jednotlivým dílčím úkolům, které nakonec zadáváme v programovém jazyce.

Velmi důležité je také zaznamenání postupu řešení ve formě poznámek. Nyní se vám zdá všechno jasné. Pokud se však dostanete k programu po delší době, nemusí vám být ihned jasné, proč jste postupovali uvedeným způsobem. Zaznamenávejte si proto maximum poznámek k řešení problému. Poznámky mohou být v programu dvojího druhu. Pokud uvedeme na začátku řádky dvě lomítka za sebou, bude systém celý řádek ignorovat a bude jej považovat za poznámku. Takto je možné například psát delší komentáře, nebo při různých pokusech můžeme zneplatnit celý řádek programu, aniž by jsme jej museli vymazávat.

Druhou možností zápisu vlastního komentáře do programu je použití složených závorek. Vše, co je v programu uvedeno mezi levou a pravou složenou závorkou je považováno za komentář a je systémem ignorováno. Tento druh poznámek je výhodné používat na konci řádku pro zápis popisu provedené činnosti řádku. Takto je možné označit jako poznámku i text uprostřed řádku, nebo několik řádků najednou.

Při zápisu programu se snažte dodržovat grafickou úpravu programu, kdy jsou těla bloků odsazena od svého záhlaví o několik znaků. Takový program je potom i na první pohled lépe čitelný a snáze se hledají případné chyby.

Systém nerozlišuje v programu malá a velká písmena. Přesto bude vhodné, pokud budete dodržovat jednotnou dále popsanou úpravu. Nejvhodnější je používat pro návěští a ukončení bloku (FOR..ELSE..ENDFOR) velká písmena. Stejně tak je vhodné psát jména základních

příkazů systému (READ, WRITELN, CONTINUE ..) velkými písmeny. Jména procedur a funkcí přebíraná z knihoven systému je vhodné zapisovat tak, aby vždy první písmeno slova bylo velké. Složeným slovem se rozumí jméno složené z několika slov, mezi nimiž nesmí být mezera. Například ImageInit, ConsoleClear apod. Stejným způsobem je vhodné pojmenovat vlastní víceznaková jména deklarovaných proměnných.

22. lekce - ladící a animační režim

V předchozí lekci jsme se seznámili s tím, jak program navrhnout a sestavit. Nikdy však není zaručeno, že vám bude program ihned pracovat správně. Chybu v programu vám může ohlásit systém v případě chybně deklarované proměnné, špatně zadaného příkazu a podobně. Avšak i v případě, že program proběhne až do konce bez vypsání chybového hlášení, neznamená to, že je program bez chyb. Chyba nemusí být v samotném zápisu programu, ale v chybném sestavení algoritmu. Protože se takové chyby špatně hledají, byl systém doplněn několika pomůckami, dovolujícím chyby v programu nalézt.

V jedné z předchozích lekcí jsme si zběžně ukázali animaci programu. Při ní se program spouští klávesou F6 případně volbou z menu Program/Animace, nebo kliknutím na ikonu s obrázkem filmu. Program přitom bude mimo vykonávání naprogramované činnosti současně zobrazovat v okně s programem řádku, kterou právě vykonává. Chod programu je současně zpomalen tak, aby bylo možné průběh animace programu sledovat. To nám umožní kontrolovat průběh programu přes jednotlivé příkazy, volání procedur a funkcí. Animaci programu máte možnost kdykoliv přerušit tak, že stisknete klávesu F9 a program bude pokračovat normální rychlostí bez zobrazování vykonávaného řádku programu. Pokud by jste chtěli animaci pouze dočasně pozastavit, můžete tak učinit kliknutím myší na ikonu s obrázkem STOP značky.

Další, velmi zajímavou možností systému vzhledem k ladění programu je schopnost průběžného vypisování hodnot proměnných. Proměnné se zobrazují ve svém samostatném okně, které otevřete z menu volbou Okna/Proměnné. Zobrazí se vám tabulka se dvěma ikonami v horní liště. Levá ikona se používá pro přidání proměnné do tabulky, pravá ikona proměnnou z tabulky zruší. Zkuste si proto napsat program, ve kterém použijete cyklus FOR..ENDFOR. Mimo řídící proměnné deklarujte i další proměnné. Jména proměnných zadejte i do tabulky pro sledování proměnných. To učiníte tak, že kliknete myší na levou ikonu, a do dotazovacího okna zadejte jméno proměnnou, kterou chcete sledovat. Po jejím zadání se zobrazí proměnné v tabulce s uvedením, že hodnota proměnné není definována. stejným způsobem zadejte i další proměnné, jejichž stav budete chtít sledovat. Při spuštění programu v režimu animace sledujte současně tabulku s hodnotami proměnných. Dokud není v programu proměnné přidělena hodnota, zobrazuje se v jejich stavu tzv. nedefinovaná náhodná veličina. Po přidělení hodnoty a její každé změně je stav hodnoty proměnné v tabulce aktualizován. Můžete proto sledovat, zda odpovídá její hodnota předpokládaným veličinám.

Mimo režimu animace, kdy jsou řádky programu vykonávány automaticky se zadanou časovou prodlevou máte možnost program krokovat sami po jednotlivých řádcích. Krokování programu po řádcích se aktivuje klávesou F7. Systém při každém stisku klávesy F7 provede pouze jeden řádek programu se současným zobrazením řádky, kterou bude provádět v následujícím kroku. Máte tak možnost sami ovládat časové prodlevy mezi prováděním jednotlivých řádek programu. Současně máte také možnost výše popsaného zobrazení hodnot proměnných v příslušném okně. Pokud se z programu volá procedura, pokračuje krokování programu ve volané proceduře. Pokud ale nechcete proceduru krokovat, stiskněte na řádku s procedurou místo F7 klávesu F8, která umožní skok do procedury přeskočit. Program v takovém případě vykoná volanou proceduru běžnou rychlostí bez zobrazování prováděných řádek programu. Obdobně to platí i pro volání funkce v programu.

Pokud budete chtít sledovat průběh programu až od určitého místa, nastavte kurzor v okně s programem na požadované místo a spusťte program funkční klávesou F4. Program se spustí běžnou rychlostí a po dosažení řádku programu, na kterém je kurzor přejde do ladícího režimu. Dále budete mít možnost pokračovat buď krokováním, animací nebo například po kontrole hodnot proměnných pokračovat běžnou rychlostí.

Výše popsaná animace programu i jeho krokování vám dovolí sledovat průběh činnosti

programu a změny hodnot proměnných. To vám umožní poznat důkladněji činnost příkazů pro cyklus a větvení programu. Krokování i animaci programu máte možnost kdykoliv přerušit s tím, že činnost programu buď pozastavíte nebo spustíte běžnou rychlostí. Přerušení se provede kliknutím myši na ikonu se STOP značkou. Pokračování programu běžnou rychlostí je možné stiskem klávesy F9. Krokování programu, jeho animaci i běžné spuštění je možné libovolně kombinovat.

Protože zůstává aktivní příkazové okno stále přístupné, máte možnost kdykoliv změnit hodnotu proměnné, vypsat si obsah proměnných, které nemáte v okně pro sledování proměnných apod. Tím dosáhnete absolutní nadvlády nad programem, což vám kompilované programy nikdy neumožní.

23. lekce - typ ARRAY - proměnná typu pole

Pokud budete potřebovat v programu více proměnných stejného typu například pro uložení výsledků výpočtů, můžete výhodně použít deklaraci typu pole (array). Pole je seskupení proměnných stejného typu, které jsou dostupné indexem pole udávající pozici prvku pole.

Jestliže budete chtít například uložit do pole výsledky náhodného generování hodnot pomocí funkce Random, které potom sečtete a vypočítáte jejich průměr, můžete použít následující program:

```
VAR
   x      : integer;
   soucet : real
   cislo  : array[1..10] OF real;
ENDVAR

PROCEDURE main
   FOR x := 1 to 10                    {pro celé pole}
       cislo[x] := Random(100);        {náhodné číslo}
       WRITELN(x:2,':',cislo[x]:10:3); {výpis hodnoty}
   ENDFOR
   soucet := 0;                        {nulování součtu}
   FOR x := low(cislo) to high(cislo)  {rozsah pomocí indexu}
       soucet := soucet + cislo[x];
   ENDFOR
   WRITELN('-------------');
   WRITELN('Součet:',soucet:6:2);
   WRITELN('Průměr:',soucet/10:6:2);   {výpočet průměru}
ENDPROC
```

V programu je deklarováno pole se jménem cislo, obsahující deset prvků typu real. K jednotlivým prvkům pole můžete přistupovat pomocí indexu pole, které se uvádí v hranatých závorkách. Například:

```
cislo[x] := Random(100);  {přiřazení do prvku pole}
WRITELN(cislo[x]:10:3);   {přístup k prvkům pole}
```

Při definici pole nemusí začínat rozsah pole od jedničky. Rozsah pole můžete přitom zjistit funkcemi Low a High, které udávají počáteční a koncový index pole. Funkce jsou uvedeny i ve výše uvedeném programu.

Vlastností polí se může použít také u práci s řetězci. Řetězec je vlastně pole znaků typu Char, aniž je nutné jej takto definovat. Proto je možné pomocí indexů pracovat přímo s jednotlivými znaky pole (řetězce):

```
s := 'OZOGAN KLONDAIK';  {počáteční hodnota řetězce}
s[7] := '*';             {na sedmou pozici vložíme *}
WRITELN(s);              {vypíše OZOGAN*KLONDAIK}
```

Pole můžeme použít i při předávání hodnot procedurám a funkcím. V tomto případě, protože není při jejich definici znám přesný počet položek pole, nazýváme uvedená pole jako otevřená. Nyní jsme již nuceni používat při výpočtech pro zjištění prvního a posledního indexu funkce Low a High. Výše uvedený příklad pro výpočet průměru upravíme tak, aby bylo možné výpočet volat jako funkci s předem nedefinovaným počtem prvků pole:

```
FUNCTION prumer(cislo : array of real) : real;
   VAR
      x      : integer;
      soucet : real;
   ENDVAR
   soucet := 0;                          {nulování součtu}
   FOR x := low(cislo) to high(cislo)  {rozsah pomocí indexu}
       WRITELN(x:2,':',cislo[x]:10:3); {výpis hodnoty}
       soucet := soucet + cislo[x];
   ENDFOR;
   WRITELN('-------------');
   WRITELN('Součet:',soucet:6:2);
   WRITELN('Průměr:',soucet/(high(cislo)-low(cislo)):6:2);
ENDFUNC

PROCEDURE main
   ConsoleClear;
   Prumer([13, 3, 5, 1]);
   WRITELN('');
   Prumer([1, 2, 3]); {zkuste vlastní hodnoty, různý počet}
ENDPROC
```

Všimněte si, že pole je deklarováno současně s deklarací funkce a nemusí se v takovém případě uvádět rozsah pole. To je uvedeno v místě volání funkce počtem předávaných parametrů. Předávané parametry musí být uvedeny v hranatých závorkách, což je příznak otevřeného pole. Jako parametry lze přitom uvádět různý počet čísel. Z hlediska systému ale převádíme vždy pouze jeden parametr typu pole, který je deklarován v podřízené proceduře nebo funkci.

Pole v systému KLONDAIK mohou být pouze jednorozměrná, čož znamená že mohou obsahovat pouze jeden index. Ten si lze jednoduše představit například jako pozice znaků v řetězci. Klasický jazyk Pascal umožňuje používat vícerozměrná pole. Položka dvourozměrného pole je určena dvěma indexy. Pokud by například šachovnice představovala dvourozměrné pole, kombinace řádek/sloupec přitom definuje požadovaný prvek pole.

Tip pro zkušené uživatele:

Aby jste obešli absenci dvourozměrného pole, můžete v případě požadavku na uložení numerických hodnot v rozsahu do 256 (typ byte) použít deklaraci jako pole, kde uvedete prvek pole typu string. Jak již bylo uvedeno, lze k řetězci přistupovat stejným způsobem jako k poli. A to je právě hledaný 'druhý rozměr'. Nejprve si prvek pole převedeme do řetězcové proměnné (první rozměr), ze které potom pomocí indexu (druhý rozměr) přistupujeme na požadované místo. K převodu znaku na číslo použijte funkci Ord, číslo na znak převede funkce Chr (bude vysvětleno v následující kapitole). Protože se jedná o složitější programátorskou konstrukci, nejsou zde uváděny další podrobnosti.

24. lekce - knihovna pro zpracování řetězců

Jak již víte, obsahuje typ string řetězec maximálně 256 znaků. Někdy je však vhodné pracovat i částí řetězce, doplňovat znaky na libovolné místo, zjistit délku řetězce a podobně. To je samozřejmě možné a takovéto funkce obsahuje knihovna pro zpracování řetězců, se kterou se nyní seznámíme.

Řetězec je sestaven z jednotlivých znaků typu char. Řetězec proto vlastně představuje pole znaků typu char. K řetězci je proto možné přistupovat jako k poli a číst, případně měnit znaky na požadovaných pozicích. Proto jsou možné následující zápisy, ačkoliv je proměnná s deklarovaná jako řetězec a ne pole, jak by se mohlo vzhledem k použitým operacím očekávat.

```
s := 'OZOGAN KLONDAIK';
WRITELN(s[4]);          {vypíše čtvrtý znak řetězce}
s[7] := '*';            {nastaví sedmý znak na hvězdičku}
```

Každý znak v řetězci má svou číselnou hodnotu, udávající pozici znaku v ASCII tabulce, ve které je umístěno všech 256 možných znaků. ASCII tabulka je pevně definována a platí i mimo systém KLONDAIK. Spodní část tabulky do pozice 125 je vždy pevně dána. Horní část tabulky již záleží na definici národního prostředí počítače (např. písmena s diakritikou). Na začátku tabulky jsou umístěny netisknutelné znaky, které se používají pro obsluhu tiskárny. Dále jsou již definovány jednotlivé znaky. Každý znak řetězce můžete proto rozložit na jednotlivé znaky a jejich hodnoty. V programu je definována funkce Ord která provádí převod hodnoty znaky na číselnou pozici v tabulce. Opačná funkce je Chr, která vrací dle zadaného čísla odpovídající znak z ASCII tabulky:

```
WRITELN(Ord('A'));  {vypíše ASCII hodnotu znak znaku 'a'};
WRITELN(Ord('a'));  {vypíše ASCII hodnotu znak znaku 'A'};
WRITELN(Chr(65));   {vypíše znak tabulky zadané hodnoty}
FOR x:= 65 to 90
    WRITE(Chr(x));  {vypíše abecedu - velká písmena}
ENDFOR; WRITELN;
FOR x:= 97 to 122
    WRITE(Chr(x));  {vypíše abecedu - malá písmena}
ENDFOR;
```

Všimněte si, že velká písmena v tabulce začínají na pozici 65 a malá písmena na pozici 97, jejich rozdíl je proto 32. To lze využít například pro převod řetězce na malá písmena.

```
s := 'ozOgaN kLonDaiK';
FOR x := 1 TO Length(s)
    IF (Ord(s[x])>=65) and  (Ord(s[x])<=90) THEN
       s := Copy(s,1,x -1)
          + Chr(Ord(s[x])+32)
          + Copy(s, x+1,Length(s)-x);
    ENDIF
ENDFOR
WRITELN(s);             {vypíše 'ozogan klondaik'}
```

Pokud potřebujete zjistit délku řetězce, použijte funkci Lenght:

```
WRITELN(Length(s));
```

Jestliže musíte z řetězce vybrat pouze jeho část, můžete použít funkci Copy, kde zadáte mimo

řetězce i počáteční pozici a délku podřetězce. Funkce Copy byla například použita ve výše uvedeném programu na změnu písmen v řetězci.

Další velmi často požadovanou funkcí je možnost vyjmout z řetězce určitou část, případně do řetězce vložit jiný řetězec. K vyjmutí podřetězce se používá funkce Delete, která ze zadaného řetězce zruší od zadané pozice zadaný počet znaků. Naopak, funkce Insert vloží na zadanou pozici do řetězce požadovaný podřetězec. Obě funkce mění současně délku řetězce.

```
s := 'Program OZOGANKLONDAIK';
Insert('*', s, 15);  {vloží znaky do řetězce}
Delete(s, 1, 8);     {vyjme znaky z řetězce}
WRITELN(s);          {vypíše: 'OZOGAN*KLONDAIK'}
```

Pokud potřebujete zjistit, zda se v řetězci nalézá hledaný podřetězec, lze použít funkci Pos. Funkce vrací pozici prvního znaku hledaného podřetězce.

```
s := 'OZOGAN KLONDAIK';
WRITELN(Pos('KLON',s)); {vrací číslo 8}
```

Velmi často budete pravděpodobně potřebovat převádět čísla na řetězce a obráceně. Jednodušší je přitom převod čísel na řetězec. Pro převod čísla typy Integer se používá funkce IntToStr. Tato funkce nedokáže převést desetinnou část čísla. K tomu se používá funkce Str, která vám umožní zadat nejen délku čísla, ale i počet desetinných míst.

```
x := 123.45;
s := IntToStr(x);  {převádí pouze celou část čísla}
WRITELN(x:6:2);    {vypíše 123.00}
s := Str(x:6:2);   {převádí včetně desetinných míst}
WRITELN(x:6:2);    {vypíše 123.45}
```

Problémy nastanou, pokud budete potřebovat převést řetězec na číslo. Pokud obsahuje řetězec opravdu pouze numerické znaky, bude vše v pořádku. Pro převod celých čísel na řetězec se používá funkce StrToInt. Pokud převádíte čísla typu Real, musíte použít funkci Val, ve které uvedete řetězec pro převod a proměnnou pro převáděnou hodnotu:

```
s := '123.45';
x := StrToInt(s);  {převede pouze celou část čísla}
WRITELN(x:6:2);    {vypíše 123.00}
Val(s,r);          {převádí řetězec do proměnné typu Real}
WRITELN(r:6:2);    {vypíše 123.45}
```

Jak již bylo uvedeno, problémy nastanou, pokud obsahuje řetězec nenumerické znaky. V takovém případě nelze použít funkce StrToInt a Val, protože ty v takovém případě vrací nulovou hodnotu převáděného čísla. Je proto nutné využít funkci Val, která vrací pozici prvního nenumerického znaku v převáděném rětězci. Pokud tedy vrátí nenulovou hodnotu, je nutné nejprve zrušit z řetězce nenumerické znaky a použít znovu funkci Val pro převod:

```
x := Val(s,r);          {zjistí pozici nečíselného znaku}
IF x > 0 THEN           {pokud nečíselný znak v řetězci}
   Delete(s, x, Length(s)-x+1); {zruší nečíselné znaky}
   Val(s,r);                    {zopakuje převod}
ENDIF
WRITELN(r:10:3);    {vypíše číslo po převodu z řetězce}
```

Výše uvedený postup jste nuceni použít v běžném jazyce Pascal. Proto byl systém KLONDAIK doplněn o funkci StrToReal, která dokáže sama rozpoznat nenumerické znaky v řetězci a ignorovat je při převodu.

25.lekce - knihovna pro práci se soubory, adresáři

Pokud jste již zkušený uživatel systému OZOGAN KLONDAIK, mohou se vám hodit funkce pro kopírování souborů a práci s adresářem na disku. Musíte si však dávat pozor, aby jste nezrušili důležitá data a soubory.

Jména souborů a adresářů se v následujících funkcích předávají ve tvaru řetězce. Pokud se soubor nachází ve stejném adresáři jako program, není nutné uvádět k němu cestu (jméno adresáře). Stejně tak nemusíte uvádět označení disku, pokud se jedná o aktuální disk.

Asi nejčastěji používanou funkcí bude kopírování souborů. K tomu nám poslouží funkce FileCopy, ve které musíte uvést jako parametr jméno zdrojového souboru a jméno cílového souboru. Funkce přitom vrací logickou hodnotu o úspěšnosti kopírování:

```
IF FileCopy('PROGRAM.IPS','PROGRAM.BAK') THEN
   WRITELN('Záložní kopie programu byla vytvořena')
ENDIF
```

Soubor se nemusí zkopírovat v mnoha různých případech. Například soubor pro kopírování nemusí existovat, na disku je málo místa a podobně. Seznámíme se proto s funkcemi, které nám zajistí zjištění informací o disku, volném prostoru a podobně. Nejprve bude vhodné zjistit, zda soubor skutečně existuje. K tomu nám poslouží funkce FileExists:

```
IF FileExists('C:\CONFIG.SYS') THEN
   WRITELN('Soubor opravdu existuje')
ENDIF
```

Dalším krokem by mohlo být zjištění velikosti souboru, aby jsme mohli následně zjistit, zda se nám kopie souboru na disk vejde. K tomu se použije funkce FileSize, která vrací velikost souboru v bajtech. Záporný výsledek signalizuje neexistenci zadaného souboru:

```
WRITELN('Velikost',FileSize('C:\AUTOEXEC.BAT')/1024:4, 'Kb');
```

Soubor můžeme kromě kopírování také přejmenovat, případně zrušit. Zde ale opatrně. Přejmenování souboru provádí funkce FileRename, ve které se musí jako parametry uvést jméno původního souboru a jméno nového souboru. Funkce vrací logickou hodnotu indikující úspěšnost operace:

```
IF FileRename('XXX.XXX','YYY.YYY') THEN
   WRITELN('Přejmenování souboru proběhlo v pořádku')
ELSE
   WRITELN('Zadaný soubor neexistuje')
ENDIF
```

Soubor lze samozřejmě také vymazat funkcí FileDelete, ve které se uvede jméno souboru pro zrušení. Funkce vrací opět logickou hodnotu udávající úspěšnost výmazu souboru:

```
IF FileDelete('YYY.YYY') THEN
   WRITELN('Soubor YYY.YYY byl zrušen');
ENDIF
```

Další skupina procedur a funkcí se zabývá přímo diskem a disketami, kdy je možné zjistit jejich velikost, volné místo, ale například také vytvářet a rušit adresáře. Pro zjištění velikosti (kapacity) disku nebo diskety se použije funkce DiskSize. Jako parametr se přitom uvádí číslo disku od A

(A=1, B=2, C=3, ..). Funkce vrací velikost v bajtech. Pokud je vrácena záporná hodnota, je tím indikována buď neexistence zařízení, případně nepřipravenost disketové mechaniky (není založena disketa):

```
x := DiskSize(1);
IF x < 0 THEN
   WRITELN('Velikost disku A: ', x/1024/1024:4:2, ' Mb');
ELSE
   WRITELN('mechanika A: není připravena');
ENDIF
```

Obdobným způsobem je možné zjistit volnou kapacitu disku funkcí DiskFree. Funkce se dá opět použít pro indikaci nepřipravenosti zařízení. V následujícím příkladě se však zjišťuje volná kapacita disku C a tak se předpokládá, že opravdu existuje.

```
WRITELN('Velikost disku C:',DiskSize(3)/1024/1024:7:3, ' Mb')
WRITELN('Volná kapacita C:',DiskFree(3)/1024/1024:7:3, ' Mb')
```

Další funkce se již zabývají přímo adresáři. Funkce GetDir zjišťuje jméno aktuálního adresáře. Parametr zadává opět číslo disku od A. Nulový parametr přitom předpokládá aktuální diskovou jednotku. Změnu pracovního adresáře provedete funkcí ChDir, ve které uvedete jméno adresáře, do kterého se chcete přepnout.

```
d := GetDir(0);
IF ChDir('\pokus') THEN
   WRITELN('Změna adresáře proběhla v pořádku');
   Chdir(d);       {přepneme se zpět do původního adresáře}
ELSE
   WRITELN('Zadaný adresář zřejmě neexistuje !');
ENDIF
```

Pokud potřebujete založit nový adresář, můžete použít funkci MkDir, kde uvedete jako parametr jméno adresáře. Zadaný adresář nesmí přitom existovat, nebo to nesmí být jméno souboru. V případě chyby vrací funkce nenulovou hodnotu. Adresář můžete naopak zrušit funkcí RmDir, kde uvedete opět jako parametr jméno adresáře pro zrušení. Adresář musí být přitom prázdný ! V případě úspěšnosti vrací funkce nulovou hodnotu.

```
IF MkDir('pokus') = 0   {pokud je adresář úspěšně založen}
   RmDir('pokus')       {tak ho opět zrušíme, ať nezavazí}
ENDIF
```

Jak již bylo uvedeno, musíte být při práci s funkcemi pro obsluhu souborů a adresářů zvlášť opatrní, protože případná chyba může mít nedozírné následky !

26. lekce - knihovna pro práci s TXT soubory

Soubory představují seskupení dat na disku počítače. Jak jsme si ukázali v předchozí lekci, soubory můžeme přejmenovat, zrušit i zkopírovat. Zatím jsme se ale neukázali, jak soubory vytvářet a jak manipulovat s jejich obsahem.

Soubory mohou mít různou vnitřní strukturu danou jejich obsahem a vznikem. My se budeme nyní zabývat pouze textovými soubory, které obsahují libovolný text. Soubor je členěn na řádky, které představují typ string. Soubory je možné zpracovávat pouze sekvenčně, což znamená postupně od začátku do konce. Není možný přímý přístup na požadovaný záznam (řádek).

Soubor, se kterým budeme chtít pracovat musíme nejprve definovat a určit jeho jméno. Soubor se definuje v bloku VAR..ENDVAR, kde se u proměnné uvede typ text. Současně je možné uvést přiřazením i jeho jméno:

```
VAR
   f : text = 'SOUBOR.TXT'  {definuje textový soubor}
ENDVAR
```

První akcí, kterou musíme vždy provést je přiřadit proměnné typu soubor konkrétní jméno souboru. To provedeme buď výše uvedenou deklarací, kde se uvede ihned i jméno souboru, nebo procedurou Assign:

```
VAR
   f : text;               {definuje proměnnou typu text}
ENDVAR
Assign(f, 'SOUBOR.TXT');  {přiřadí proměnné konkrétní soubor}
```

Další akcí, kterou musíme se souborem provést je jeho otevření. Pokud budeme chtít založit nový soubor, otevřeme jej funkcí Rewrite. Existující soubor se otevře pro zápis funkcí Append. Pokud budete chtít soubor pouze číst, otevře se soubor funkcí Reset. Uvedené funkce vrací v případě úspěšnosti otevření souboru nulovou hodnotu. Číst ze souboru, případně zapisovat do něj je možné pouze v případě že byl soubor úspěšně otevřen.

Pokud budete chtít změnit typ prováděné operace (zápis/čtení), musíte soubor nejprve uzavřít a znovu otevřít podle požadované práce se souborem. Nelze otevřít soubor, který je již používán.

Zápis do souboru se provádí příkazem WRITE, nebo WRITELN. Přitom se musí uvést vždy jako první parametr soubor, kam má být výstup proveden. Například:

```
WRITELN(f, 'OZOGAN KLONDAIK');
```

Pokud by jste neuvedli výstupní soubor, byl by výstup proveden do textového výstupního okna. Při výstupu do souboru můžete kombinovat příkazy WRITE a WRITELN. Při použití příkazu WRITE se provede výstup do souboru bez ukončení řádku. Další výstup bude potom prováděn na stejný řádek. Řádek bude ukončen příkazem WRITELN. Stejně jako u výstupu do textového výstupního okna.

Soubor musíte na konci vždy uzavřít, jinak by nebyl soubor dále přístupný. Uzavření souboru se provádí funkcí Close. Uvádíme nyní kompletní příklad výstupu informací do textového souboru:

```
PROCEDURE main
   VAR
      f : text = 'SOUBOR.TXT'  {definuje textový soubor}
```

```
      x : byte;
   ENDVAR

   ConsoleClear;               {vymaže textové výstupní okno}
   IF (Rewrite(f) = 0) THEN    {otevře soubor pro zápis}
      WRITELN(f,'OZOGAN KLONDAIK');
      FOR x := 1 to 10
          WRITELN(f, x:2, ':', 10/x:10:3);
      ENDFOR;
      Close(f);                {uzavře soubor}
   ENDIF;
ENDPROC;
```

Číst ze souboru můžete po jeho otevření funkcí Reset. Čtení se provádí příkazem READLN, ve kterém musíte uvést jako parametr mimo proměnné, do které se má načtení uložit i soubor, ze kterého se má číst. Pokud by jste tak neučinili, proběhlo by čtení proměnné ze vstupního okna. Pro indikaci konce souboru slouží funkce Eof, která je pravdivá při jeho čtení v okamžiku dosažení konce souboru. Přečtení a vypsání souboru vytvořeného v předchozím příkladě může být naprogramováno například následujícím způsobem:

```
PROCEDURE main
   VAR
      f : text;
      s : string;
   ENDVAR

   Assign(f, 'SOUBOR.TXT'); {přiřadí proměnné soubor}
   ConsoleClear;            {vymaže textové výstupní okno}
   IF (Reset(f) = 0) THEN   {otevře textový soubor}
      WHILE (NOT Eof(f))    {dokud není konec souboru}
         READLN(f,s);       {přečte řádek souboru}
         WRITELN(s);        {vypíše řádek na obrazovku}
      ENDWHILE;             {konec cyklu}
      Close(f);             {uzavře soubor}
   ENDIF;
ENDPROC
```

Tip pro zkušené uživatele:

Příkaz READLN čte ze souboru jeden řádek, který ukládá do zadané proměnné. Typ proměnné může být libovolný číselný, nebo řetězcový. Pokud budete chtít ale načíst do numerické proměnné řádek, ve kterém jsou uloženy nenumerické údaje, bude nahlášena chyba. Pokud proto neznáte přesnou strukturu souboru, doporučuje se načítat řádek do proměnné typu string. Až potom je možné dál rozložit načtený řádek ze souboru na jednotlivé elementy. Využijete přitom již dříve probrané funkce pro práci s řetězci.

27. lekce - typ RECORD - strukturované proměnné

Jak již bylo uvedeno, musí se všechny hodnoty v programu ukládat do proměnných definovaného typu. Systém přitom obsahuje několik předdefinovaných základních typů. Tyto základní typy je možné rozšířit o další odvozené. Nově definované typy se mohou skládat nejen z předdefinovaných základních typů, ale i z dříve definovaných vlastních typů.

Vlastní typy se většinou sdružují do tzv. strukturovaných údajů které se nazývají záznamy (anglicky Record). To je výhodné například pro definici data, kde sdružíme položky datum, měsíc a rok do jednoho záznamu. Obdobně je možné definovat nový typ pro adresu nebo například osobní údaje (jméno, narození, adresa). To nám následně umožní přistupovat ke každé složce záznamu samostatně. Pro definici nového typu se používá v programu blok TYPE..ENDTYPE. Takto by jste například nadefinovali vlastní typ pro uložení data a adresy, následně potom ihned osobní záznam, kde námi definované typy použijete:

```
TYPE
   datum = RECORD
      den   : byte;
      mesic : byte;
      rok   : byte;
   ENDRECORD

   adresa = RECORD
      ulice : string;
      mesto : string;
   ENDRECORD

   osoba  = RECORD
      jmeno   : string;
      narozen : datum;
      bydliste: adresa;
      telefon : string;
   ENDRECORD
ENDTYPE
```

Záznamy se musí definovat bloku TYPE..ENDTYPE, na začátku programu. Mají proto vždy globální platnost. Záznam může obsahovat v sobě další, dříve definovaný záznam. V programu potom můžete uvést deklaraci proměnné, ve které uvedete nově definovaný typ:

```
VAR
   seznam : osoba;
ENDVAR
```

Pro přístup k jednotlivým polím typu záznam musíte použít pro definici jména název záznamu a název pole záznamu oddělené tečkou:

```
seznam.jmeno := 'OZOGAN';
seznam.bydliste.ulice := '1.máje 97';
seznam.bydliste.mesto := 'Liberec';
WRITELN(seznam.jmeno);
```

Záznamy nám umožní nejen rozčlenit uložené údaje na jednotlivé elementy, ze kterých se skládají, ale současně i analyzovat podrobněji vlastnosti určitého celku. Víme například, že datum se

skládá ze dne, měsíce a roku. Stejně tak by jsme mohli definovat, že den se skládá z hodin, minut a sekund. Všem těmto elementům se z hlediska programování říká objekty. Objekty nám přitom umožňují popsat dokonale nejen vše kolem nás, ale také struktury údajů v programech.

Rozšířené struktury dat v podobě záznamů použijeme v následující lekci pro definici tabulek pro uložení údajů s možností vyhledávání a zpracování pomocí klíčů.

28. lekce - typ TABLE (záznamy)

Systém OZOGAN KLONDAIK umožňuje pracovat pouze se soubory typu text. Textové soubory je možné zpracovávat pouze sekvenčně, to znamená postupně od začátku do konce. Při tomto způsobu není možné vyhledávání podle klíčů. Proto byl systém doplněn o několik procedur a funkcí s možností zpracování záznamů pomocí vyhledávacích indexů. To Vám umožní zpracovávat záznamy s možností vyhledávání požadovaných záznamů pomocí klíčů. K tomuto účelu je nutné definici záznamu doplnit o označení klíčových položek slovem KEY. Tabulka zákazníků by mohla mít například následující deklaraci:

```
TYPE
   ZakaznikRec  = RECORD
      KEY firma : string;
      mesto     : string;
   ENDRECORD
ENDTYPE
```

V záznamu jsou deklarovány dvě položky. Jedna z nich, označená slovem KEY slouží pro vyhledávání požadovaného záznamu. Dále musíme v programu deklarovat proměnnou typu tabulka, ve které budou zákazníci uloženi. Tabulka bude mít přitom strukturu danou výše deklarovaným typem ZakaznikRec. Stejného typu bude i proměnná Zakaznik, sloužící k veškeré komunikaci s tabulkou. Z programu budeme k tabulce přistupovat vždy pouze pomocí této proměnné a příslušných procedur.

```
VAR
   ZakaznikTab : TABLE of ZakaznikRec; {definice tabulky}
   Zakaznik    : ZakaznikRec;          {záznam o zákazníkovi}
ENDVAR
```

Nejprve potřebujeme do tabulky zapsat postupně všechny požadovaná data. Do proměnné Zakaznik proto naplníme údaje jednoho zákazníka a ty potom následně zapíšeme do tabulky. K tomu nám poslouží procedura WriteRecord, která uloží do zadané tabulky nový, zadaný záznam. Při tvorbě tabulek mějte na paměti, že celá tabulka je uložena v operační paměti jejíž kapacita je omezená. Nelze proto vytvářet velmi rozsáhlé tabulky.

```
Zakaznik.firma := 'OZOGAN';
Zakaznik.mesto := 'Liberec';
WriteRecord(ZakaznikTab, Zakaznik);

Zakaznik.firma := 'SH PLUS';
Zakaznik.mesto := 'Liberec';
WriteRecord(ZakaznikTab, zakaznik);
```

Výše uvedeným způsobem naplníme tabulku. Musíme si však uvědomit, že pokud používáme strukturu záznamu s klíčem, nedovolí systém nejednoznačnost klíčů a proto v případě zápisu záznamu s klíčem, který již existuje, bude tento nahrazen novým záznamem. Tím je zajištěna jednoznačnost vyhledávacích klíčů.

Při veškeré komunikaci s tabulkou používáme proměnnou Zakaznik, ve které nastavíme požadovaný vyhledávací klíč, který předáme tabulce procedurou SetKeysFromRecord. Až potom lze provádět další akce.

Pokud potřebujete vyhledat a načíst z tabulky požadovaný záznam, zadejte nejprve do proměnné Zakaznik požadovaný vyhledávací klíč a následně pomocí procedury SetKeysFromRecord nastavte

požadovaný vyhledávací klíč v tabulce na zadanou hodnotu. Funkcí ReadRecord potom zjistíte, zda záznam s požadovaným klíčem existuje. Pokud ano, přesune se do proměnné Zakaznik obsah celé datové struktury zákazníka k dalšímu použití:

```
Zakaznik.firma := 'SH PLUS';
SetKeysFromRecord(ZakaznikTab, Zakaznik);
IF (ReadRecord(ZakaznikTab, Zakaznik)) THEN
   WRITELN(Zakaznik.firma, Zakaznik.mesto);
ELSE
   WRITELN('Záznam ', Zakaznik.firma, ' nebyl nalezen');
ENDIF
```

Při požadavku na zrušení záznamu z tabulky musíte opět zadat nejprve do proměnné Zakaznik požadovaný vyhledávací klíč a následně pomocí procedury SetKeysFromRecord nastavit požadovaný vyhledávací klíč v tabulce na zadanou hodnotu. Funkcí DeleteRecord bude následně v případě jeho nalezení záznam zrušen:

```
Zakaznik.firma := 'SH PLUS';
SetKeysFromRecord(ZakaznikTab, Zakaznik);
IF DeleteRecord(ZakaznikTab) THEN
   WRITELN('Záznam ', Zakaznik.firma, ' byl zrušen');
ENDIF
```

Tabulkou seřazenou podle klíče je možné sekvenčně procházet záznam po záznamu a provádět s každým záznamem zadané operace. K tomu slouží procedura ForEachRecord, ve které zadáte jméno tabulky, jméno záznamu, který slouží pro komunikaci s tabulkou a jméno procedury, která bude záznamy tabulky zpracovávat. Procedura je potom volána pro každý záznam a v proměnné, kterou jste uvedli jako parametr procedury ForEachRecord je obsažen záznam z tabulky.

```
PROCEDURE PrintRecord;     {procedura volaná pro každý záznam}
   WRITELN(Zakaznik.firma, Zakaznik.mesto);
ENDPROC

PROCEDURE main
...
ForEachRecord(ZakaznikTab, Zakaznik, 'PrintRecord');
...
ENDPROC
```

Vytvořenou tabulku máte možnost uložit do souboru na disku a také ji zpětně načíst pro další zpracování. Tabulka se uloží na disk procedurou SaveTable. Načtení tabulky do paměti se provede funkcí LoadTable. Pokud proběhne načtení tabulky v pořádku, je vrácena hodnota True. V opačném případě (neexistence souboru, málo paměti) je vrácena hodnota False.

```
IF LoadTable(ZakaznikTab,'ZAKAZNIK.DTA') THEN
   WRITELN('Tabulka byla načtena');
   ...                              {provedení změn v tabulce}
   SaveTable(ZakaznikTab, 'ZAKAZNIK.DTA');
ENDIF
```

29. lekce - Zprávy uživateli, dotazy a výběry

Občas budete potřebovat oznámit uživateli z programu nějakou zprávu, vyžádat si odpověď typu ANO/NE, vybrat některou volbu z několika nabízených a podobně. Pokud pomineme již dříve probraný příkaz READ, obsahuje systém několik procedur a funkcí, které jsou vhodné pro tvorbu dotazů uživateli.

Příkaz READLN umožňuje zadat pouze krátkou, jednořádkovou odpověď. Vyžadujete-li zadání několikařádkové odpovědi, můžete použít funkci Answer, která vám umožní zadat řetězec o délce maximálně 254 znaků rozdělený na řádky. Parametr funkce ve formě řetězce se zobrazí nad vstupním polem pro zadání textu jako nadpis. Pokud je funkce ukončena stiskem tlačítka OK, předá funkce jako výsledek hodnotu True. Současně s tím je v předdefinované systémové proměnné IT vrácen obsah zadaného textu k dalšímu požití.

```
IF (Answer('Zadejte adresu:')) THEN
   WRITELN('Vaše adresa je:');
   WRITELN(IT);
ENDIF
```

Tip pro zkušenější uživatele:
Pokud vypíšete obsah proměnné IT do textového výstupního okna, bude již formátován dle zadaných řádků. Ve skutečnosti se ale jedná stále o jeden řetězec, jehož jednotlivé řádky jsou odděleny uvnitř řetězce znaky pro 'nový řádek' a 'návrat vozu'. Pokud budete proto chtít rozložit zadaný text na jednotlivé řádky, budete muset použít funkce pro nalezení znaků Chr(13)+Chr(10), které ukončují každý řádek a funkce Copy pro vyseparování jednoho řádku z řetězce, kde uvedete zjištěnou pozici konce řádku..

Další možností komunikace programu s uživatelem je výpis zprávy se zobrazením tlačítek pro odpověď uživatele. Pomocí funkce MessageBox můžete vypsat uživateli zprávu v samostatném oznamovacím okně, kde bude zobrazena mimo vaší zprávy i ikona charakterizující zprávu a příslušná tlačítka pro odpověď od uživatele. Máte možnost zadat také titulek okna. Funkce vyžaduje zadání tří parametrů: text zprávy, titulek okna a příznak udávající zobrazená tlačítka a vybranou ikonu. Pro příznak jsou v systému definovány konstanty, které usnadní zadávání hodnot. Příslušný příznak vypočítáte součtem příznaků pro požadovaná tlačítka a zobrazenou ikonu. Příznak můžete zadávat buď přímo číselnou hodnotou, nebo vypsáním jména konstanty:

```
příznak pro zobrazení tlačítek:
-------------------------------------------------------------
hodnota konstanta            význam
-------------------------------------------------------------
   0  MB_OK                  zobrazí pouze tlačítko OK
   1  MB_OKCANCEL            zobrazí tlačítka OK a STORNO
   2  MB_ABORTRETRYIGNORE    zpráva obsahuje tlačítka PŘERUŠIT,
                            ZNOVU a OPAKOVAT
   3  MB_YESNOCANCEL         zobrazí tlačítka ANO, NE a STORNO
   4  MB_YESNO               zobrazí tlačítka ANO a NE
   5  MB_RETRYCANCEL         zobrazí tlačítka ZNOVU a STORNO
-------------------------------------------------------------
příznak pro zobrazení ikony:
-------------------------------------------------------------
hodnota konstanta            význam
-------------------------------------------------------------
```

```
  16    MB_ICONSTOP          zobrazí v okně ikonu STOP
  32    MB_ICONQUESTION      zobrazí v okně ikonu s otazníkem
  48    MB_ICONEXCLAMATION   zobrazí v okně ikonu vykřičníku
  64    MB_ICONINFORMATION   zobrazí v okně ikonu Info
-------------------------------------------------------------

MessageBox('Výpočet ?','Dotaz',MB_YESNO+MB_ICONQUESTION);
{nebo}
MessageBox('Výpočet ?','Dotaz',36);  {4 + 32 = 36}
```

Funkci MessageBox můžete volat i jako proceduru, kdy nezáleží na odpovědi uživatele. Například informace o ukončení akce. Při volání funkce MessageBox je vrácena hodnota nula, pokud není dost paměti pro vytvoření okna se zprávou. Jinak vrací příznak stisknutého tlačítka podle následujících hodnot:

```
-------------------------------------------------------------
hodnota  konstanta   význam
-------------------------------------------------------------
1        IDOK        bylo vybráno tlačítko OK
2        IDCANCEL    bylo vybráno tlačítko STORNO
3        IDABORT     bylo vybráno tlačítko PŘERUŠIT
4        IDRETRY     bylo vybráno tlačítko ZNOVU
5        IDIGNORE    bylo vybráno tlačítko IGNOROVAT
6        IDYES       bylo vybráno tlačítko ANO
7        IDNO        bylo vybráno tlačítko NE
-------------------------------------------------------------
```

Pokud budete chtít, aby uživatel vybral jednu z nabízených možností, můžete použít funkce Choose, ve které zadáte jako parametry nápis, který se zobrazí nad polem pro výběr voleb a seznam nabízených voleb. Pokud budete chtít, aby byla na počátku nastavena jiná, než první volba, musíte tuto volbu uvést na začátku seznamu a potom ještě jednou na příslušném místě dle požadovaného pořadí položek pro výběr. Funkce vrací jako výsledek pořadí vybrané volby s číslováním od nuly. Současně obsahuje předdeklarovaná systémová proměnná IT obsah vybrané volby. Pokud je výběr přerušen stiskem tlačítka 'Zrušit', vrací funkce zápornou výslednou hodnotu.

```
IF (Choose('Vyberte den v týdnu:', 'sobota',  'pondělí',
           'úterý', 'středa', 'čtvrtek', 'pátek',
           'sobota', "neděle') >= 0 ) THEN
   MessageBox('Vybral jste si: ' + IT,
              'Oblíbený den v týdnu', MB_OK);
   WRITELN('Váš oblíbený den je: '+ IT);
ELSE
   WRITELN('Výběr byl přerušen', 'výsledek', MB_OK);
ENDIF
```

Výše uvedenou funkci Choose můžete použít pouze v případě, že znáte již v době psaní programu přesný počet voleb pro výběr. Pokud bude počet voleb znám až v době vykonávání programu, máte možnost použít obdobnou funkci ChooseOption. Nejprve však budete muset aktivovat výběrové pole procedurou ClearChooseBox. Dále potom procedurou AddChooseItem zadáte jednotlivé volby pro výběr. Plnění můžete provést například výběrem dat z tabulky apod. Následující příklad používá pro zjednodušení naplnění přímo z programu. Výběr požadované volby od uživatele se uskuteční funkcí ChooseOption, ve které uvedete text, který se zobrazí nad výběrovým

polem a text volby, která má být nastavena.

Funkce vrací jako výsledek pořadí vybrané volby s číslováním od nuly. Současně obsahuje předdeklarovaná systémová proměnná IT obsah vybrané volby. Pokud je výběr přerušen stiskem tlačítka 'Zrušit', vrací funkce zápornou výslednou hodnotu.

```
ClearChooseBox
AddChooseItem('leden');
AddChooseItem('únor');
AddChooseItem('březen');
...
AddChooseItem('prosinec');
IF (ChooseOption('Vyberte měsíc v roce:', 'leden') >= 0) THEN
   MessageBox('Vybral jste si:' + IT, 'Výběr', MB_OK);
   WRITELN('Váš oblíbený měsíc je: '+ IT);
ENDIF
```

Znalost tvorby dotazů s využitím výše uvedených procedur a funkcí umožní tvořit vzhledné a zajímavé programy. Dalším stupněm a vrcholem systému je možnost tvorby formulářů v samostatných oknech, kterými se budeme zabývat v následující kapitole.

30. lekce - tvorba a používání formulářů

Pro možnost komunikace s uživatelem můžete použít vlastní definované formuláře, které mohou obsahovat statický text, editační textové pole, tlačítko vypínače a příkazové tlačítko. Formulář obsahuje vždy v oblasti pravého horního rohu tlačítka OK a Zrušit.

Procedury a funkce pro definici a obsluhu formulářů obsahují většinou mnoho parametrů. Podrobnosti jsou proto uvedeny podrobněji v nápovědě k programu, případně v popisu jazyka INTER-PASCAL.

Formulář se musí nejprve nadefinovat procedurou CreateForm, ve které se zadává jméno formuláře, titulek okna s formulářem a rozměry formuláře. Dále se nadefinují na plochu formuláře další editační objekty. Rozměry a vzdálenosti se přitom zadávají v bodech. Nejprve se udávají souřadnice levého horního okraje a potom šířka a výška formuláře nebo objektu na formuláři. Volitelně je možné zadat jméno procedury, která se provede při inicializaci formuláře a může obsahovat například nastavení počátečních hodnot editovaných na formuláři.

Pořadí zadávání objektů na formulář je důležité pro možnost pozdějšího přístupu k jednotlivým objektům definovaných na formuláři. Každý objekt na formuláři je očíslován od jedničky v pořadí jejich definice. Číslo objektu se potom používá například ke změně hodnot, viditelnosti objektu a podobně. Číslo objektu je vytvářeno jako výsledek funkce tvorby objektu na formuláři a je vráceno jako identifikátor ID.

Statický text se na formulář zadává funkcí AddStatic, ve které se uvádí jako parametr jméno formuláře, text záhlaví a souřadnice. Funkce vrací identifikátor ID udávající pořadí vytvoření objektu na formuláři. Statický text vypisuje na formuláři nadpisy, apod. Maximální délka textu je 30 znaků.

Editační box sloužící k editaci textové položky se definuje na formuláři funkcí AddEditBox. Parametry funkce udávají jméno formuláře, vstupní a výstupní proměnnou a souřadnice editační položky. Poslední volitelný parametr udává jméno procedury, která se použije v okamžiku změny obsahu editačního pole. Funkce vrací identifikátor ID určující pořadí vytvoření objektu na formuláři.

Pokud budete chtít editovat na formuláři položku typu boolean s hodnotami True a False (ANO/NE), použijte funkci AddCheckBox. V parametrech se zadává jméno vstupní a výstupní logické proměnné, text uvedený u přepínače a souřadnice umístění souřadnice na formuláři. Poslední volitelný parametr udává jméno procedury, která se použije v okamžiku změny obsahu přepínače. Funkce vrací identifikátor ID udívající pořadí vytvoření objektu na formuláři.

Posledním typem objektu je příkazové tlačítko, které umožňuje spouštět na povel uživatele programu definovanou proceduru. Příkazové tlačítko se definuje na formulář funkcí AddPuschButton, ve které se v parametrech uvede jméno formuláře, jméno procedury, která se spustí po stisknutí tlačítka, text na tlačítku a souřadnice umístění tlačítka na formuláři. Funkce vrací identifikátor ID udávající pořadí vytvoření objektu na formuláři.

Výše uvedené funkce slouží pouze k definici formuláře a neprovádí žádnou viditelnou činnost. Teprve až funkce FormDialog zobrazí nadefinovaný formulář ve tvaru dialogového okna a je zahájena jeho editace. Jako parametr funkce se uvádí jméno formuláře. Funkce vrací logickou hodnotu udávající způsob ukončení editace formuláře. Pokud byla editace ukončena stiskem tlačítka OK, vrací funkce výsledek True, jinak vrací hodnotu False.

Výše uvedené funkce již zajišťují samy o sobě dostatečné možnosti pro editaci formuláře. Pokud budete chtít využívat i podřízené procedury uvedené v definici editačního boxu, přepínacího a příkazového tlačítka, můžete ovládat nastavení hodnot editovaných proměnných, skrytí a zobrazení objektů formuláře, případně i jejich znepřístupnění k editaci. To vše na základě využití

identifikátoru ID, který je vracen jako výsledek definice objektu na formuláři.

Většinou budou akce probíhat tak, že pokud je v definici objektu uvedena volitelná procedura volaná v okamžiku ukončení editace objektu, budete mít možnost v této proceduře například zkontrolovat a nastavit hodnoty proměnných libovolného objektu, zpřístupnit je k editaci, případně nastavit viditelnost objektu. Tím získáte mnoho možností programové obsluhy formuláře.

Nastavení hodnoty libovolné proměnné podle hodnoty editačního boxu se provede funkcí GlgGetControlText. Obdobně se přidělí logické proměnné nová hodnota funkcí GlgGetControlCheck dle stavu přepínače. Jako parametr funkce se uvádí identifikátor ID objektu udávající pořadí jeho definice na formulář.

Nastavení hodnoty editačního boxu podle stavu proměnné se provede funkcí GlgSetControlText. Obdobně se nastaví stav přepínače na novou hodnotu funkcí GlgGetControlCheck. Jako parametr funkce se uvádí identifikátor ID objektu udávající pořadí jeho definice na formulář.

Pokud budete chtít některou editační položku znepřístupnit, můžete k tomu použít funkce DlgDisableControl. Zpřístupnění editační položky se provede funkcí DlgEnableControl. Zjištění přístupnosti editační položky se provede funkcí DlgIsControlEnabled. Jako parametr funkce se uvádí identifikátor ID objektu udávající pořadí jeho definice na formuláři.

Jestliže budete muset některou editační položku zneviditelnit, můžete k tomu použít funkce DlgHideControl. Zviditelnění editační položky se provede funkcí DlgShowControl. Zjištění viditelnosti editační položky se provede funkcí DlgIsControlVisible. Jako parametr funkce se uvádí identifikátor ID objektu udávající pořadí jeho definice na formulář.

Podrobné informace o funkcích sloužící k editaci formulářů včetně popisu parametrů naleznete v příslušných odkazech nápovědy. Nyní uvedeme na závěr pouze krátkou ukázku vytvoření a editace jednoduchého formuláře.

```
PROCEDURE DisplayForm
   VAR
      pol1_imp : string = 'vstup 1'; pol2_imp : string = 'vstup 2';
      pol3_imp : string = 'vstup 3'; pol4_imp : boolean = true;
      pol5_imp : boolean = false; pol1_out : string;  pol2_out : string;
      pol3_out : string;  pol4_out : boolean; pol5_out : boolean;
   ENDVAR
   IF (formDialog('Form1')) THEN
      messageBox(pol2_out, 'obsah druhé editační položky', MB_ICONINFORMATION + MB_OK);
   ENDIF;
ENDPROC

PROCEDURE pushProc
  IF (DlgIsControlEnabled(2)) THEN
     DlgDisableControl(2);
     MessageBox('Editační Box 1 znepřístupněn','Zpráva uživateli', MB_ICONINFORMATION+MB_OK);
  ELSE
     DlgEnableControl(2);
     MessageBox('Editační Box 1 zpřístupněn', 'Zpráva uživateli', MB_ICONINFORMATION+MB_OK);
  ENDIF
  FormDialog('Form2')
ENDPROC

PROCEDURE hideCtl
   IF (dlgIsControlVisible(7)) THEN
```

```
      dlgHideControl(7);
   ELSE
      dlgShowControl(7);
   ENDIF
ENDPROC

PROCEDURE updateStatic
   VAR
      t : string = 'první položka';
   ENDVAR
   t := dlgGetControlText(2)
   dlgSetControlText(10, 'obsah položky 1 = ' + t);
ENDPROC

PROCEDURE main
   createForm   ('Form2', 'Prázdný formulář', 300, 300, 350, 100);
   createForm   ('Form1', 'Test formuláře', 100, 100, 270, 250);
   addStatic    ('Form1', 'položka 1', 10, 30, 50, 20);
   addEditbox   ('Form1', 'pol1_imp', 'pol1_imp', 10, 50, 150, 20, 'UpdateStatic');
   addStatic    ('Form1', 'položka 2', 10, 72, 50, 20);
   addEditbox   ('Form1', 'pol2_imp', 'pol2_out', 10, 92, 150, 20);
   addStatic    ('Form1', 'položka 3', 10, 114, 50, 20);
   addEditbox   ('Form1', 'pol3_imp', 'pol3_out', 10, 134, 150, 20);
   addCheckbox  ('Form1', 'pol4_imp', 'pol4_out', 'Check &Box', 10, 165, 90, 20);
   addCheckBox  ('Form1', 'pol5_imp', 'pol5_out', '&Ukryj Box', 10, 185, 90, 20, 'HideCtl');
   addPushButton('Form1', 'PushProc', 'zde &stiskněte', 160, 165, 100, 24);
   addStatic    ('Form1', '', 10, 205, 140, 20);
   displayForm;
ENDPROC
```

31. lekce - Rozšířené možnosti formulářů

V předchozí lekci jsme si probrali základní informace o používání formulářů v systému KLONDAIK. Nyní si ukážeme několik rozšiřujících vlastností, které nám umožní mnohem dokonalejší využití formulářů.

Na formuláři je možné kromě již dříve uvedených objektů definovat funkcí AddIcon i zobrazení ikon (soubory *.ICO) a bitmap (soubory *.BMP). Tlačítka je možné také definovat funkcí AddSpeedButton, s možností zobrazení bitmapy místo textu. Pro lepší grafickou úpravu formuláře je vhodné používat funkci AddFrame, která definuje na formuláři rámeček. Pokud by jste potřebovali měnit za chodu programu pozici zobrazené ikony, máte možnost tak učinit procedurou MoveIcon. Obdobně můžete změnit procedurou ChangeIcon zobrazovanou ikonu, případně bitmapu. Stejně tak je možné procedurou MoveButton změnit pozici tlačítka na formuláři. Uvedené schopnosti jsou využitelné a vhodné hlavně na programování her na ploše formuláře, kdy je možné měnit z programu zobrazení ikon a jejich pozice.

Pokud vám vadí, že standardní formulář zobrazuje uzavírací tlačítka formuláře "OK" a "Zrušit", můžete použít proceduru FormActionKey, která dokáže kromě ukrytí uvedených tlačítek snímat navíc i kódy stisknutých kláves. Pokud přitom nadefinujete proceduru, která má reagovat na stisk klávesnice, máte možnost komfortně ovládat další návazné akce. To je opět použitelné i na programování her, kdy můžete reagovat například na stisk kurzorových kláves.

Další zajímavou možností je schopnost snímat pozici stisknutého tlačítka myši na ploše ikony nebo bitmapy. K tomu je nutné uvést při definici ikony (bitmapy) funkcí AddIcon jména proměnných, do kterých se bude pozice stisknutého tlačítka myši ukládat. Uvedená vlastnost je použitelná opět například při programování her. Využít se dá ale také pro snímání souřadnic zobrazených bitmap - například map a podobně.

32. lekce - Spouštění externích a DLL programů

V předchozích lekcích jsme si ukázali a probrali mnoho možností, které systém KLONDAIK umožňuje. Ještě dříve, než se seznámíme se způsobem zpracování databází, tabulek a grafů si ukážeme, jak je možné doplnit do programu mnoho dalších užitečných schopností.

Určitě vás již napadlo, že by bylo příjemné mít možnost spouštět přímo ze systému další externí programy. Ty by mohly provádět například rozsáhlé akce, pro jejichž použití není systém KLONDAIK vhodný. Uvedený požadavek je do systému zabudován dokonce s dvěma principiálně odlišnými způsoby řešení.

Voláním funkce RunPrg máte možnost spustit přímo ze systému externí EXE program, který provede požadovanou akci. Může to být přitom například textový editor, nebo také program pro zpracování požadovaných výpočtů a podobně. Je nutné však přitom dbát na specifika systému Windows. Ne všechny programy je možné takto spustit bez problémů. Například programy pro systém MS-DOS nebo rezidentní programy mohou způsobit někdy těžko předvídatelné situace.

Protože systém Windows pracuje v režimu multitaskingu, může být spuštěno najednou několik programů, které běží souběžně. Při jejich spouštění funkcí RunPrg je proto možné zadat, zda má systém KLONDAIK čekat na dokončení zadaného programu a až potom pokračovat ve vykonávání další řádky programu. Čekání na ukončení spuštěného programu je nutné například dodržet pokud budete volat ze systému KLONDAIK postupně několik externích programů zpracovávající uzavřenou celkovou úlohu:

```
RunPrg('PROGRAM1.EXE', '', '', 1, true );{čeká na ukončení}
RunPrg('PROGRAM2.EXE', '', '', 1, false);{nečeká na ukončení!}
```

Další možností spouštění externích programů je určena především pro zkušené odborníky. Pokud vám nevyhovují pevně definované knihovny procedur a funkcí systému KLONDAIK, máte možnost si jednoduše doplnit do systému vlastní knihovny funkcí pomocí DLL knihoven. K tomu je nutné znát samozřejmě přesnou definici DLL knihovny a proto je zde uveden pouze rámcový popis. Odborníci, pro které jsou uvedené možnosti určeny si jistě poradí.

Před použitím funkcí DLL knihovny je musíte v programu nejprve definovat. V definici musíte uvést jména funkce z knihovny, parametry a název knihovny. Definici musíte umístit na začátku programu.

Následující příklad využití DLL knihovny používá knihovny dodávané spolu se systémem KLONDAIK, které slouží pro práci s tabulkami. V příkladu je volána funkce pro zjištění maximálního rozsahu sloupců a řádků tabulky:

```
external Function SSMaxcol : word ; module "vtssdll.dll";
external Function SSMaxRow : word ; module "vtssdll.dll";

PROCEDURE main
   writeln(ssMaxCol:12:5);
   writeln(ssMaxRow:12:5);
ENDPROC
```

33. lekce - Spouštění programů jako makra

Výše uvedené možnosti spouštění externích EXE a DLL programů jsou vhodné k naprogramování posloupnosti volání programů vykonávající kompletní úlohu v dávce ve stylu *.BAT souborů v systému MS-DOS. Vzniknou tak výkonná makra systému Windows dovolující mimo spouštění externích programů navíc i inteligentní rozhodování dalšího postupu na základě výsledku provedených akcí.

V programu spuštěném jako makro můžete například ve vstupním formuláři zadat počáteční data, uložit je do souboru, který již dále zpracuje spuštěný externí program. Další možností je například načtení hodnot zařízení zapojeného na port počítače, následné zpracování externím programem a vytištění výsledků. Data je možné si mezi systémem KLONDAIK a externími programy předávat v souborech, tabulkách nebo databázích. Sami jistě naleznete vhodné možnosti a výhodnosti uvedeného použití systému KLONDAIK.

Odladěné a odzkoušené programy systému KLONDAIK lze spouštět i mimo jeho vývojové prostředí. To lze provést tak, že si definujete ve Windows asociaci souborů s příponou *.IPS na systém KLONDAIK. Pokud potom kliknete v manažeru souborů na program vytvořený v systému KLONDAIK, bude spuštěna ihned interpretace programu bez zobrazení vývojového prostředí. Pokud takovýto program ukončíte příkazem Quit, bude ukončen nejen spuštěný program, ale současně i vývojové prostředí systému. Uvedená vlastnost není v demoverzi systému dostupná.

Do dalších verzí systému KLONDAIK se připravuje (zatím nelze) možnost napsaný program zakódovat do nečitelného tvaru a ten potom následně spouštět i mimo vývojové prostředí KLONDAIKu ve stylu výše uvedených výkonných maker Windows. Výhodou bude nejen ochrana programu před neoprávněnými zásahy do jeho činnosti, ale hlavně možnost provozovat takovýto program přímo pod demoverzí systému KLONDAIK. Tím se stane demoverze runtime modulem vytvořených programů bez nutnosti vlastnit plnou verzi programu. Zakódované programy spuštěné v demoverzi systému nebudou samozřejmě omezeny funkčností demoverze (délka programu, časové omezení apod.).

34. lekce - Co je to databáze, z čeho se skládá

Databáze je uspořádaná skupina informací, které mají k sobě definovaný vztah. Všechna data, mající k sobě vztah tvoří jeden záznam. Záznam se skládá z polí. Každé pole je jedním ze zásadních údajů záznamu a u všech záznamů se opakuje. Na příklad v adresáři tvoří jedna adresa jeden záznam. Tento záznam se skládá z různých dat uložených v polích (např. ulice, město, telefon), která mají vzájemný vztah - patří jedné adrese. V adrese je například jedním z polí poštovní směrovací číslo, které se u všech záznamů jako údaj opakuje. U každého záznamu však může mít různou hodnotu. Databáze mohou být například adresáře, telefonní seznamy, seznamy zaměstnanců, ceník zboží a podobně. Tedy jakékoliv souhrny údajů, které se ve stálé skladbě opakují.

Všechna data pro jeden objekt databáze se nazývají záznam, nebo také věta databáze. Například v databázi zákazníků obsahuje každý záznam všechna data jednoho zákazníka. Záznam databáze představuje v databázovém okně jeden řádek.

Každá z informací uvnitř záznamu se nazývá položka záznamu a obsahuje pro každý záznam jeden konkrétní údaj. Například v databázi adres je to jméno, ulice, PSČ apod. V databázovém okně představuje položka databáze vždy jeden sloupec. V prvním, barevně odlišeném řádku sloupce je přitom uvedeno jméno položky databáze.

Položky databáze jsou pojmenovány a jejich jména jsou spolu s dalšími vlastnostmi databáze uloženy ve struktuře databáze. Struktura databáze tedy definuje, jaké pole bude databáze mít, jak budou velká a jaký typ dat do nich bude možné zadávat. Struktura databáze je přitom fyzicky její součástí a je uložena vždy na jejím začátku.

Struktura databáze obsahuje následující údaje:
Jméno pole - může obsahovat maximálně 10 znaků, to je písmen a číslic včetně podtrhávací pomlčky. Jméno položky databáze nesmí obsahovat mezeru a musí začínat písmenem.

Typ pole - určuje typ uložených dat a možnost prováděných operací s daty. Typ pole může být řetězcový (typ C), numerický (typ N), datumový (typ D), logický (typ L) a memo pole (typ M) - poznámka libovolné délky).

Délka pole - určuje maximální počet znaků v poli. U numerických položek se udává délka včetně znaménka (mínus), desetinné tečky a desetinných míst.

Počet desetinných míst - určuje u numerických položek počet desetinných míst za desetinnou tečkou.

35. lekce - Ovládání databázového okna

Na začátku programu je databázové okno vždy prázdné. Nejprve musíte požadovanou databázi otevřít. To provedete nejlépe kliknutím na příslušnou ikonu v horní liště databázového okna, případně procedurou DbfUse.

Pokud je otevřena databáze, jsou její data zobrazena na celé ploše databázového okna. Data jsou zobrazována v mřížce, kdy přestavují řádky mřížky záznamy databáze a sloupce mřížky prezentují položky databáze. V horním, barevně odlišném prvním řádku mřížky jsou zobrazeny názvy položek databáze. Zobrazená mřížka se v databázových jazycích nazývá BROWS.

Aktuální databázový záznam je v mřížce barevně zvýrazněn. Ve stejné řádce zcela vlevo je současně na stejném řádku v úzkém sloupci zobrazen formou trojúhelníku ukazatel databáze. Kurzorovými klávesami máte možnost se přesouvat v databázi na libovolnou položku kteréhokoliv záznamu. Navíc máte možnost použít pro rychlý přesun v databázi následující klávesy:

Home	- přesun na první položku záznamu (první zleva)
End	- přesun na poslední položku záznamu (první zprava)
PageUp	- přesun o obrazovku nahoru
PageDown	- přesun o obrazovku dolů
Ctrl+Home	- přesun na první záznam databáze
Ctrl+End	- přesun na poslední záznam databáze

V horní části databázového okna je umístěn tzv. navigátor, který slouží také k přesunu mezi záznamy databáze. Navigátor obsahuje čtyři ikony sloužící k přesunu na první záznam, záznam o jeden vpřed, záznam o jeden vzad a na poslední záznam. Pátá ikona v navigátoru se zobrazuje pouze v případě, že máte nastavenou možnost editace a slouží k označení záznamu ke zrušení, případně k obnovení platnosti záznamu.

V dolní části databázového okna se ve stavovém řádku zobrazuje číslo aktuálního záznamu spolu s celkovým počtem záznamů v databázi. Uvedené počty jsou odděleny lomítkem.

Nevyhovuje-li vám zobrazení datové mřížky, můžete změnit myší její vzhled. Velikost položek máte možnost změnit jednoduše tak, že uchopíte myší čáru mezi názvy položek v horní části datové mřížky a posunete ji požadovaným směrem. Tím dojde k rozšíření, případně k zúžení sloupce položky databáze. Obdobným způsobem lze změnit pořadí zobrazování položek v datové mřížce. K tomu musíte uchopit myší jméno položky umístěné v horní řádce mřížky a přesunout jej na požadované místo. Uvedenými úpravami se databáze žádným způsobem nemění a pokud ji znovu otevřete, zobrazí se v původním tvaru.

Po otevření databáze je standardně nastaveno, že není možné záznamy z databázového okna měnit. Možnost změny dat nastavíte z PopUp menu, které se zobrazí po kliknutí pravého tlačítka myši na ploše datové mřížky. Kliknutím na volbu 'Možnost editace' se přepíná možnost editace záznamů databáze.

Přidávat nové záznamy do databáze lze také pouze pokud máte nastavenou možnost editace databáze. Pro přidání nového záznamu stiskněte klávesu Insert. Zobrazí se nový, prázdný záznam, u něhož můžete zapsat obsah položek. Záznam se ihned zařadí na konec databáze. Pokud je databáze indexována, zařadí se nový záznam podle nastaveného indexu.

Současně s možností editace je zpřístupněna i možnost rušení záznamů z databáze. Zrušení záznamu databáze se provede kliknutím na příslušnou ikonu navigátoru. Aktuální záznam databáze není zrušen ihned. Je pouze označen ke zrušení. Záznam je možné obnovit opětovným kliknutím na ikonu v navigátoru. Informace o tom, zda je aktuální databázový záznam určen ke

zrušení je zobrazována v dolní části databázového okna v jeho stavovém řádku. V takovém případě je za číslem aktuálního záznamu zobrazen text [Deleted]. Záznamy určené ke zrušení je možné z databáze definitivně zrušit pouze voláním procedury DbfPack.

Pokud chcete, aby se záznamy označené ke zrušení nezobrazovaly, máte možnost nastavit uvedený požadavek v popup menu, které se zobrazí po stisku pravého tlačítka myši na ploše databázového okna. Volba "Zobrazovat zrušené záznamy" působí jako přepínač a pokud je volba označena, budou se záznamy označené ke zrušení v databázovém okně zobrazovat. Současně budou přístupné i z programu. Pozor na to, že pokud změníte nastavení, budou aktuálně zobrazené záznamy na ploše databázového okna obsahovat původní zobrazení. Až po jejich novém načtení do okna se budou zobrazovat správné záznamy. Proto se doporučuje po změně nastavení zobrazování zrušených záznamů provést v databázovém okně klávesou PageUp nebo PageDown přechod na jiné záznamy. Nastavení přístupnosti záznamů označených ke zrušení je možné provést i procedurou DbfSetDeleted. Nastavená volba přístupnosti zrušených záznamů je po novém spuštění systému obnovena tak, jak byla nastavena při posledním ukončení systému.

Vpravo od navigátoru je umístěna ikona pro tisk databáze. Databázi je možné tisknout pomocí tzv. reportů, což jsou předdefinované popisy tiskových sestav. Ty je možné přitom velmi jednoduše měnit vizuálně myší na ploše návrhu sestavy. Dále lze zadávat skupinování sestavy, součtování a podobně. Podrobněji se návrhem sestav zabývá samostatná lekce kurzu.

V pravé horní části databázového okna je umístěn výběrový box pro zadání způsobu řazení záznamů. Pokud je databáze indexována, obsahuje box názvy indexů a vy máte možnost jejich výběrem změnit způsob řazení záznamů dle požadovaného indexu s možností rychlého vyhledávání požadovaného záznamu. Podrobnější informace o indexování databáze budou uvedeny v následujících lekcích.

36. lekce - Navigace v databázi

V předchozí lekci jsme si ukázali, jak listovat v databázi v databázovém okně. Nyní se seznámíme s obdobnými možnostmi přímo z programu.

Každý záznam v databázi je očíslován dle pořadí zápisu do databáze. První záznam má proto pořadové číslo jedna, poslední zapsaný záznam má pořadové číslo shodné s celkovým počtem záznamů v databázi. Číslo aktuálního záznamu včetně celkového počtu záznamů v databázi je uveden ve spodní části databázového okna v jeho stavovém řádku. Celkový počet záznamů databáze vrací funkce DbfReccount, číslo aktuálního záznamu vrací funkce DbfRecNo. Pokud by jste chtěli vypsat uvedený zápis z programu, můžete použít následujícího zápisu:

```
WRITELN(IntToStr(DbfRecNo)+"/"+IntToStr(DbfReccount));
```

Pokud potřebujete přejít na požadované číslo záznamu, můžete tak učinit voláním procedury DbfGo, ve které uvedete jako parametr číslo požadovaného záznamu. Pro přechod na první záznam databáze použijte volání procedury DbfGoTop. Skok na konec databáze provedete procedurou DbfGoBottom. Jestliže budete chtít listovat v databázi pouze po jednom záznamu, můžete použít volání procedury DbfSkip, ve které uvedete jako parametr počet záznamů o které se má přesunout ukazatel v databázi. Kladné číslo uvádí počet záznamů směrem na konec databáze, záporné číslo znamená skok směrem na začátek databáze.

Pokud by jste se pokoušeli přejít voláním procedury DbfSkip před první záznam bude nastaven ukazatel databáze na první záznam databáze. Současně však bude nastaven příznak začátku databáze, který je přístupný voláním funkce DbfBof. Obdobně, pokud by jste se pokoušeli provést skok v databázi za jeho konec, bude nastaven příznak konce databáze, který je přístupný voláním funkce DbfEof. Využitím výše uvedených funkcí můžete realizovat procházení databáze od jejího počátku na konec (nebo obráceně):

```
IF DbfUse("CENIK") then          {pokud se otevřela databáze}
   DbfGoTop                      {skok na začátek databáze}
   WRITELN("Začátek databáze");
   WHILE not DbfEof              {dokud není konec databáze}
      WRITELN(IntToStr(DbfRecNo)+"/"+IntToStr(DbfReccount));
      ...                        {další akce se záznamem}
      DbfSkip(1);                {skok na další záznam}
   ENDWHILE
   WRITELN("Konec databáze");
ENDIF
```

Použitím výše uvedené kostry programu máte možnost zpracovávat jednotlivé záznamy databáze. V následující lekci se proto již seznámíme s možnostmi přímého zpracování databázových záznamů (čtení a záznam položek databáze).

37. lekce - Zadávání výrazů v jazycích xBase

Pokud budeme potřebovat zpracovat obsah databázových položek, budeme muset používat ve výrazech přímo syntaxi jazyků xBase. Jsou to jazyky a systémy, které jsou přímo určeny pro zpracování databází stejného typu, který se používá i v systému KLONDAIK. Jedná se o Dbase, FoxPro a Clipper, které prodělaly dlouholetý vývoj. Stejné výrazy v syntaxi jazyků xBase budeme muset používat i my při nastavování podmínek, filtrů a indexování.

Výrazy v jazycích xBase mohou vracet výsledky logického, řetězcového nebo numerického typu. Pro přímé vyhodnocování obsahu databází lze použít volání funkcí DbfEvalLog pro získání logického výsledku, DbfEvalNum pro získání numerického výsledku a DbfEvalStr pro získání řetězcového výsledku. Parametrem uvedených funkcí je vždy řetězec obsahující výraz zadaný v syntaxi jazyků xBase.

Výrazy jazyků xBase se zadávají jako řetězce, které obsahují přímo požadovaný výraz. Ve výrazech se používají funkce a jména položek databáze spojená operátory. Ve výrazech se při zápisu jmen funkcí a názvů databázových položek nerozlišují velká a malá písmena. Pro spojování výrazů se používá následujících operátorů:

.AND. a zároveň
.OR. nebo
.NOT. negace

Nejjednodužší jsou logické výrazy, ve kterých většinou zjišťujeme obsah položek databáze. Pokud zadáte chybný výraz, bude vrácena hodnota False.

```
VAR
   log : boolean;
ENDVAR

{cena je větší, než 100}
log := DbfEvalLog("CENA>100");

{cena je větší než 100 a zároveň menší nebo rovna 200}
log := DbfEvalLog("CENA>100.AND.CENA<=200");

{cena je menší než 100 nebo obsah položky TYP je 'DISK'}
log := DbfEvalLog("CENA<100.OR.TYP='DISK'");

{první tři písmena obsahu položky NAZEV jsou 'FAX'}
log := DbfEvalLog("SUBSTR(NAZEV,1,3)='FAX'");
```

Jak jste si jistě všimli, lze jedním vyhodnocením kontrolovat obsah několika položek, navíc odlišného typu. Pozor si budete muset dávat, pokud budete chtít zadat obsah řetězce. Použít budete muset kombinaci znaků apostrof a uvozovka. Viz výše uvedený příklad.

Podobným způsobem lze vyhodnocovat i výrazy vracející numerické výsledky. Pokud zadáte chybný výraz, bude vrácena nulová hodnota.

```
{hodnota položky CENA}
WRITELN(DbfEvalNum("CENA"))

{výpočet ceny včetně DPH}
```

```
WRITELN(DbfEvalNum("CENA/100*(100+DPH)"):10:2)

{délka databázové položky NAZEV}
WRITELN(DbfEvalNum("LEN(NAZEV)"))

{pozice výskytu znaku 'A' v položce NAZEV}
WRITELN(DbfEvalNum("AT('A',NAZEV)"))
```

Lze také samozřejmě vyhodnocovat výrazy obsahující výsledek typu řetězec. Pokud zadáte chybný výraz bude vrácen prázdný řetězec.

```
{hodnota položky NAZEV}
WRITELN(DbfEvalStr("NAZEV"))

{prvních deset znaků položky NAZEV}
WRITELN(DbfEvalStr("SUBSTR(NAZEV,1,10)"))

{obsah položky NAZEV převedený na velká písmena}
WRITELN(DbfEvalStr("UPPER(NAZEV)"))

{cena včetně DPH převedená na řetězec}
WRITELN(DbfEvalStr("STR(CENA/100*(100+DPH),12,2)"))
```

Jak již bylo uvedeno, vrací chybně uvedený výraz svou implicitní hodnotu. Stane se tak v případě, že je uveden například chybný výraz xBase, neexistující jméno položky databáze a podobně. Implicitní hodnota je vrácena pokud použijete volání výše uvedených funkcí a nebudete mít v aktuální datové oblasti otevřenou žádnou databázi. Pokud budete chtít prověřit správnost zadaného výrazu, můžete použít funkci DbfEvalTest, která výraz nejen prověří, ale současně také vrací typ použitého výrazu.

38. lekce - Čtení a zápis dat databáze

V předchozích lekcí kurzu jsme se seznámili s tím, co je to databáze a jak se pohybovat v záznamech databáze. Umíme již také editovat položky databáze, avšak pouze za použití databázového okna. Nyní si proto ukážeme, jak lze přímo z programu nejen měnit obsah položek databáze, ale jak je také číst a dále zpracovávat.

Jak jsme se již dříve seznámili, mohou být položky v databázi řetězcové, numerické, logické a datumové. S každým typem položky se přitom musí v programu pracovat odlišným způsobem. Speciálním typem položky je typ Memo, což je poznámka libovolné délky, se kterou systém KLONDAIK zatím nepracuje. Nejjednodušší způsob jak zjistit typ položky je zobrazit si z databázového okna dialog pro nastavení filtru databáze, ve kterém je přesný popis databáze zobrazen.

S jednou možností čtení obsahu položek databáze jsme se již seznámili v předchozí lekci kurzu. Je to použití vyhodnocování výrazů, kdy zadáme jako výraz například pouze jméno požadované proměnné. Další možností je použití funkce DbfReadDat pro čtení datových položek, DbfReadLog pro čtení logických položek, DbfReadNum pro čtení numerických položek a DbfReadStr pro čtení řetězcových položek. Jako parametr volání funkce se použije jméno čtené položky databáze. V současné verzi systému KLONDAIK není možné číst obsah položek typu Memo. Dále přitom platí, že pokud zadáte chybné jméno položky, případně použijete pro čtení položky funkci jiného typu, bude vrácena implicitní hodnota.

Pøíklad ètení položek databáze:

```
WRITELN(DbfReadDat("DATUM"));
WRITELN(DbfReadNum("CENA"):12:2);
WRITELN(DbfReadStr("NAZEV"));
```

Ještě dříve, než se seznámíme s možností zápisu hodnot do položek databáze, ukážeme si, jak vytvořit v databázi nový, prázdný záznam. Používá se k tomu procedura DbfAppendBlank, která doplní do databáze jeden nový, prázdný záznam a nastaví na něj ukazatel databáze. Následně potom máme možnost naplnit jednotlivé položky databáze požadovanými hodnotami.

Pro zápis do databáze se používají opět procedury rozdělené podle typu položek databáze. Pro zápis do datových položek se použije procedura DbfWriteDat, pro zápis do logických položek se použije procedura DbfWriteLog, pro zápis do numerických položek se použije procedura DbfWriteNum a pro zápis do řetězcových položek se použije procedura DbfWriteStr. V současné verzi systému KLONDAIK není možné zapisovat do položek typu Memo.

Pøíklad zápisu do položek databáze:

```
DbfWriteDat("DATUM", "24/12/1997");
DbfWriteDat("DATUM", DbfEvalStr("DTOC(DAY())"));
DbfWriteNum("CENA",  100);
DbfWriteNum("CENA",  DbfReadNum("CENA")*1.1);
DbfWriteStr("NAZEV", "KLONDAIK");
DbfWriteStr("NAZEV", DbfEvalStr("SUBSTR(NAZEV,1,10)"));
```

Při zápisu do databáze musíte dávat pozor na to, že ztratíte nenávratně předchozí obsah položek a pokud použijete k experimentům přímo databáze používané například vámi používaným ekonomickým programem, můžete ztratit i mnohem více. Nepoužívejte proto ve vlastním zájmu v systému KLONDAIK databáze využívané jinými systémy.

39. lekce - Nastavení filtru, podmínky xBase

Pokud budete potřebovat omezit počet záznamů zobrazených v databázi, máte možnost nastavit filtr na zobrazení dat. To znamená, že určíte podmínky, za kterých mají být záznamy databáze přístupné. Můžete například nastavit, že mají být z ceníku přístupné pouze záznamy, u nichž je cena menší než 100. Lze také nastavit sloučení několika podmínek. Například cena je větší než 100 a zároveň menší než 200. S takto filtrovanou databází můžete provádět další akce, například tisknout sestavu, do které však budou zahrnuty pouze záznamy vyhovující zadanému filtru. Databáze zůstává přitom beze stále beze změny a nevyhovující záznamy jsou stále v databázi obsaženy. Po zrušení filtru budou dále běžně přístupné.

Filtr na databázi můžete nastavit voláním procedury DbfSetFilter, kdy se zadává přímo podmínka filtru v syntaxi jazyků xBase. Příklad zadání výše uvedené podmínky:

```
DbfSetFilter("CENA>100.AND.CENA<200");
```

Pokud by jste chtěli nastavit filtr pro cenu menší než 100 nebo větší než 200, museli by jste zadat:

```
DbfSetFilter("CENA<100.OR.CENA>200");
```

V podmínce filtru se používají přímo funkce v syntaxi jazyků xBase (databázové jazyky Dbase, FoxPro, Clipper), se kterými jsme se již seznámili v předchozích lekcích kurzu.

Další možností nastavení filtru je použití dialogového okna, který vyvoláte volbou 'Nastavit filtr' z PopUp menu zobrazeného po kliknutí pravým tlačítkem myši na ploše databázového okna. Zobrazí se dialogové okno, ve kterém je v levé části zobrazena struktura databáze s uvedením typu položek a jejich délek. Okno slouží ke snadnému zadání podmínky filtru bez nutnosti vypisování všech údajů. To je nahrazeno kliknutím myší na nabízenou variantu.

Ve střední části okna je uveden seznam funkcí v syntaxi xBase použitelných k zadání podmínky filtru. V pravé části okna jsou nahoře zobrazeny tlačítka s možnými operacemi s daty (rovná se, menší, větší, menší nebo rovno, větší nebo rovno, nerovná se apod.). V pravé střední části obrazovky jsou zobrazena tlačítka pro spojování podmínek (.AND., .OR., .NOT.). Ve spodní části okna je zobrazována podmínka filtru s možností její editace z klávesnice. V pravé dolní části obrazovky jsou umístěna tlačítka pro uzavření okna se zadáním požadované akce. Kliknete-li na tlačítko zrušit, nebude filtr nastaven. Kliknitím na tlačítko 'Nastavit filtr' aktivujete v případě jeho správného sestavení zadaný filtr, data budou omezena dle zadané podmínky.

Požadovaný výraz pro nastavení filtru máte možnost zadat přímo z klávesnice ve spodní části dialogového okna. Máte také možnost použít tlačítek pro pohodlné zadávání jednotlivých elementů výrazu pro nastavení filtru. V případě požadavku zadání výše uvedeného filtru by jsme postupovali následujícím způsobem:

1) Po kliknutí myší na položku CENA v levé části okna se zobrazí položka ve spodní části okna, kde je podmínka sestavována.

2) Klikneme myší na symbol > který je umístěn v pravé horní části okna. Symbol se zobrazí i v sestavované podmínce.

3) Z klávesnice dopíšete hodnotu 100.

4) Kliknete v pravé střední části obrazovky na tlačítko .AND. symbolizující spojení podmínky.

5) Obdobným způsobem doplníte druhou část filtru, to je CENA<200

6) Filtr aktivujete kliknutím na tlačítko Nastavit.

Po aktivaci filtru se zadaná podmínka zobrazí v horní části databázového okna. V takovém případě se zobrazují pouze data vyhovující zadané podmínce. Pokud chcete podmínku deaktivovat, vyvolejte opět okno pro zadání filtru a uzavřete jej tlačítkem Zrušit. Filtr bude deaktivován a budou opět zobrazována všechna data.

Filtr máte samozřejmě možnost nastavit i přímo z programu. Pokud například potřebujete zvýšit v ceníku ceny, které jsou včetně DPH větší než 1000,- Kč můžete použít následující program využívající následující program. Pro ty z vás, kteří neví co je to DPH bude stačit, vysvětlení, že se jedná o zvýšení ceny v procentech, jehož hodnota je uvedena v položce DPH. Jedná se o daň z přidané hodnoty. V programu nejprve nastavíme filtr a následně již zpracujeme každý záznam. Nevyhovující záznamy jsou díky nastavenému filtru nepřístupné.

```
IF DbfUse("CENIK") then          {pokud se otevřela databáze}
   DbfSetFilter("(CENA/100*(100+DPH))<1000");
   {nastavit filtr pokud cena s daní menší než 1000 Kč}
   DbfGoTop                      {skok na začátek databáze}
   WHILE not DbfEof              {dokud není konec databáze}
      DbfWriteNum("CENA",DbfReadNum("CENA")*1.1);
      {zvýšení ceny o 10 %}
      DbfSkip(1);                {skok na další záznam}
   ENDWHILE
   DbfSetFilter("");             {zrušit nastavený filtr}
ENDIF
```

V následující lekci se seznámíme i s dalšími možnostmi řešení výše uvedeného úkolu.

40. lekce - Sekvenční prohledávání databáze

Pokud potřebujeme zobrazit z databáze pouze určité záznamy podle zadané podmínky, můžeme nastavit na databázi filtr, se kterým jsme se seznámili v předchozí lekci kurzu. Filtrování databáze je však výhodné pouze při prohlížení databáze v okně. Pokud je ale u rozsáhlé databáze použit filtr, který vybere pouze malou část databáze, může být zobrazování vyhovujících záznamů pomalé. Pokud proto nepožadujete použít filtr pro výběr vyhovujících záznamů v databázovém okně, ale přímo v programu, je výhodnější použít dále popsaného způsobu.

Použijeme zadání úkolu z předchozí lekce, kdy máme zvýšit v ceníku ceny, které jsou včetně DPH menší než 1000,- Kč. Pro ty z vás, kteří neví co je to DPH bude stačit, vysvětlení, že se jedná o zvýšení ceny v procentech, jehož hodnota je uvedena v položce DPH. Jedná se o daň z přidané hodnoty. V programu procházíme databází záznam po záznamu od jejího počátku a kontrolujeme námi zadanou podmínku. Pokud záznam vyhovuje, provedeme požadované akce. V našem případě zvýšení ceny.

```
IF DbfUse("CENIK") then          {pokud se otevřela databáze}
   DbfGoTop                      {skok na začátek databáze}
   WHILE not DbfEof              {dokud není konec databáze}
      IF DbfEvalNum("CENA/100*(100+DPH)")<1000 then
         {pokud cena s daní menší než 1000 Kč}
         DbfWriteNum("CENA",DbfReadNum("CENA")*1.1);
         {zvýšení ceny o 10 %}
      ENDIF
      DbfSkip(1);                {skok na další záznam}
   ENDWHILE
ENDIF
```

Jak jistě sami postřehnete, je výše uvedený program je značně neefektivní. Prochází všechny záznamy v databázi a kontroluje splnění zadané podmínky. Pouze pokud je splněna, provede zvýšení ceny. Neefektivita vzniká tím, že se prochází sekvenčně (postupně) všechny záznamy přímo v programu. Mnohem výhodnější je přece ponechat výběr požadovaných záznamů přímo systému. Berte proto výše uvedené řešení pouze jako ukázku složitějšího zpracování databáze záznam po záznamu.

K vyhledání záznamu podle zadané podmínky lze výhodně použít volání funkce DbfLocate. Ta sama prochází databázi a hledá záznam podle zadané podmínky. Hledání je proto mnohem rychlejší, než výše uvedené sekvenční prohledávání databáze. Pro hledání dalšího vyhovujícího záznamu se potom používá volání funkce DbfContinue, která hledá další vyhovující záznam. Ačkoliv obě uvedené funkce vrací přímo číslo nalezeného záznamu v databázi, používá se častěji indikace nalezení konce databázového souboru.

```
IF DbfUse("CENIK") then       {pokud se otevřela databáze}
   DbfLocate("(CENA/100*(100+DPH))<1000");
   WHILE not DbfEof           {dokud není konec databáze}
      DbfWriteNum("CENA",DbfReadNum("CENA")*1.1);
      {zvýšení ceny o 10 %}
      DbfContinue;             {hledání dalšího záznamu}
   ENDWHILE
ENDIF
```

Až dosud jsme pracovali s databází, která byla seřazena podle pořadí zápisu záznamů do databáze. V následující lekci se seznámíme s možnostmi, jak databázi seřadit a prohlížet

podle zadané položky. To nám následně i umožní rychlé vyhledávání.

41. lekce - Třídění databáze - indexy

Databáze je při svém vytváření řazena podle pořadí pořizování záznamů. Pořadí záznamů je tedy chronologické, dle času pořízení. Pro práci s databází však budete potřebovat častěji řazení podle zadaných údajů. Proto je možné databázi indexovat. To nám umožní databázi seřadit vzestupně podle zadaného údaje. Znaková pole podle abecedy, numerická pole podle velikosti čísel, datumová pole chronologicky dle data.

Při indexování databáze se vytváří tzv. indexový soubor, ve kterém jsou umístěny informace o indexování databáze. Indexový soubor je v systému KLONDAIK vždy stejného jména, jako má databáze, má však pro rozlišení příponu souboru *.CDX. Indexový soubor, pokud existuje, se otevírá vždy automaticky spolu s databází. V jednom indexovém souboru mohou být přitom uloženy informace o několika navzájem nezávislých indexech. Jedná se o tzv. vícenásobné indexové soubory typu CDX.

Zda je otevřená databáze indexována poznáte snadno v databázovém okně, kde je v případě nalezení indexů možno ve výběrovém boxu vybrat existující index. Po výběru indexu je databáze ihned řazena podle požadovaného indexu. V takovém případě nemusí souhlasit pořadí vět zobrazované ve stavovém řádku databázového okna. Bude-li některý index databáze aktivní, bude databáze řazena ne podle čísla záznamu, ale vzestupně podle zadaného indexu. Pokud zadáte přechod na začátek nebo konec databáze, nebude proveden přechod podle čísla záznamu, ale vždy podle zvoleného indexu!

Z programu máte možnost nastavit požadovaný index voláním funkce DbfSetOrder, kde uvádíte jako parametr jméno indexu, případně za použití funkce DbfTagName pořadové číslo indexu:

```
DbfSetOrder("NAZEV");
DbfSetOrder(DbfTagArea(2));
```

Pokud zadáte chybný název indexu, případně neexistující číslo indexu, bude databáze řazena podle pořadových čísel.

Nový index můžete vytvořit voláním funkce DbfIndexTag, ve které zadáte jméno indexu a výraz, ze kterého se index skládá. Jméno indexu může mít maximálně osm znaků a je možné si jej libovolně zvolit. Nejlépe je uvádět jej podle obsahu indexu. Výraz, podle kterého je databáze indexována může obsahovat výraz v jazyce xBase udávající položky databáze pro indexování. V nejjednodušším případě stačí uvést pouze název položky, podle které má být databáze indexována.

Nejjednodušší je tvorba indexů pro řetězcové položky, protože systém indexuje vnitřně vše jako rětězce. Není také problém při požadavku na spojení několika položek do indexu:

```
DbfIndexTag("NAZ", "NAZEV");
DbfIndexTag("XXX", "TYP+NAZEV");
```

Pokud by položka NAZEV obsahovala texty s velkými i malými písmeny, bylo by jejich řazení podle výše uvedeného indexu na první pohled trochu nelogické. Získali by jsme například následující pořadí: 'aa', 'aZ', 'AA'. Je to proto, že velká a malá písmena mají při třídění rozdílný význam. Pokud však vytvoříme index, ve kterém převedeme všechna písmena v řetězci na stejnou velikost, bude již vše v pořádku ('aa', 'AA', 'aZ'). Při tvorbě indexů můžeme proto používat s výhodou libovolné výrazy v syntaxe jazyků xBase, které nám umožní mnohá kouzla s indexy:

```
DbfIndexTag("NAZ", "UPPER(NAZEV)");
```

Jednoduchá je také tvorba indexů pro numerické položky. Pokud vytváříme index pouze pro

jednu položku, stačí zadat pouze její jméno. V případě že budeme chtít indexovat databázi podle několika položek, musíme pro index vytvořit za použití výrazů v syntaxi xBase pro indexování řetězec:

```
DbfIndexTag("CENA", "CENA");
DbfIndexTag("XXX", "TYP+STR(CENA,10,2)");
```

Je dokonce možné indexovat tzv vypočtené položky. To je takové, které v databázi ve skutečnosti neexistují a jejich hodnota je vypočtena z existujících položek. V následujícím příkladě bude ceník indexován podle konečné ceny včetně DPH:

```
DbfIndexTag("CENA","CENA/100*(100+DPH)");
```

Trochu složitější je indexování datumových položek. Je to proto, že ačkoliv se tváří jako řetězec, vzniknul by jejich prostým indexováním podobný problém jako u velkých a malých písmen v řetězcích. Tentokrát by se datové položky indexovaly v pořadí podle dnů, měsíců a roků. Protože je ale nutné indexovat nejprve v rámci roku, potom měsíce a nakonec dne, budeme muset použít opět funkci v syntaxi xBase, která zajistí převod data do vhodné formy pro indexování:

```
DbfIndexTag("DAT", "DTOS(DATUM)");
```

Indexy je možné tvořit buď funkcí DbfIndexTag, jak bylo výše uvedeno, nebo je možné využít dialogového okna, které vyvoláte z PopUp menu zobrazeného po kliknutí pravým tlačítkem myši na ploše databázového okna. Platí přitom stejné zásady uvedené pro tvorbu indexů funkcí DbfIndexTag.

V následující lekci se seznámíme s tím, že kromě řazení záznamů v databázi lze indexy použít i pro velmi rychlé vyhledávání záznamů v databázi.

42. lekce - Hledání v databázi dle indexů

Výhodou indexování databáze je kromě řazení podle indexů i možnost rychlého vyhledávání podle zadaného klíče. Vyhledávat podle indexů je přitom možné pouze podle indexovaných položek databáze.

Pokud budete chtít vyhledat například v adresáři konkrétní jméno, musíte si nejprve nastavit index podle jména. Následně můžete vyhledat požadované jméno buď voláním funkce DbfSeek, případně PopUp volbou z databázového okna. Bude-li zadané jméno v databázi existovat, přesune se ukazatel databáze na záznam s uvedeným jménem. Pokud zadané jméno v databázi neexistuje, přesune se ukazatel databáze automaticky na nejbližší následující záznam. Tedy na následující v řazení dle abecedy, případně podle velikosti čísla.

Při vyhledávání podle indexů nemusíte zadávat celý klíč, stačí pouze jeho začátek. V takovém případě systém hledá podle zadaného řetězce a nastaví databázový ukazatel na první vyhovující záznam. Pokud tedy zadáte pro vyhledávání například písmeno "H", bude nalezen a zobrazen první záznam začínající uvedeným písmenem.

Funkce DbfSeek vrací logickou informaci o tom zda bylo hledání dle klíče úspěšné. Navíc je při každém hledání v databázi do příznaku DbfFound nastavena informace o úspěšnosti hledání. Příznak lze testovat voláním funkce DbfFound pro provedení libovolných následných akcí.

Vyhledávání v databázi se provádí přesně podle zadaného klíče včetně respektování velkých a malých písmen v databázi i vyhledávacím klíči. Proto je možné zadat při tvorbě indexu jeho převod na velká písmena jak bylo uvedeno v předchozí lekci. Pokud potom při zadávání klíče provedete také jeho konverzi na velká písmena, bude odpovídající záznam nalezen vždy bez ohledu na zadání malých a velkých písmen.

```
IF DbfUse("CENIK") then       {pokud se otevřela databáze}
   IF DbfSetOrder("NAZ") = 0 then
      {pokud neexistuje index NAZEV, tak jej vytvoříme}
      DbfIndexTag("NAZ", "UPPER(NAZEV)");
   ENDIF
   DbfSetOrder("NAZ");      {nastavíme index podle názvu}
   DbfSeek("FAX");          {hledáme text FAX}
   IF not DbfFound          {pokud není nalezen}
      WRITELN("Záznam nebyl nalezen");
   ENDIF
ENDIF
```

Používání indexování databází má mnoho výhod a předností. Bohužel, existují i některé záporné vlasti, se kterými vás nyní seznámíme. Pokud by jste vytvořili v databázi mnoho indexů, mohlo by být jejich zpracování pomalejší. Nevýhodou je současně i prostor, který zabírají indexy v souborech na disku počítače.

Pokud není nutné používat databázi seřazenou podle indexů, vypněte jejich nastavení funkcí DbfSetOrder. Zpracování bude probíhat rychleji. Někdy se mohou indexy (soubor na disku) porušit a v takovém případě je vhodné provést procedurou DbfReindex jejich obnovu.

Není možné zrušit pouze jeden index z databáze. Lze zrušit pouze všechny indexy najednou. To je možné provést zrušením souboru, který má jméno shodné s databází, příponu má *.CDX.

43. lekce - Rušení záznamů v databázi

Jak jsme se již dříve seznámili, lze záznamy z databáze velmi jednoduše zrušit. Protože je však zrušení záznamů řešeno dvoustupňově, seznámíme se s ním nyní podrobněji. Nejdříve však jedno velmi důležité upozornění a varování. Nezasahujte jakýmkoliv způsobem do databází, které jsou využívány jiným systémem. To znamená nejen nyní zmiňované rušení záznamů, ale také jakákoliv změny dat.

Z databázového okna lze záznam zrušit pouze pokud máte nastavenou možnost editace záznamů. Tato možnost se nastavuje v PopUp menu, které se zobrazí pokud kliknete pravým tlačítkem myši na ploše databázového okna. Možnost editace se přepíná v horní řádce menu. Pokud je editace možná, je tato volba v menu zatržena a v horní liště databázového okna se zobrazí vedle šipek navigátoru nová ikona sloužící k výmazu a obnovení záznamů.

Pokud jsme nastaveni v databázovém okně na záznam, který není označen na zrušení a klikneme na ikonu výmazu, bude aktuální záznam označen ke zrušení. Poznáme to tak, že se ve stavovém řádku databázového okna zobrazí vedle čísla záznamu indikace "[Deleted]". Pokud by jsme klikli na ikonu výmazu znovu, byla by opět obnovena plná platnost databázového záznamu. Výmazová ikona tedy slouží jako přepínač. Platný záznam označí ke zrušení, u označeného záznamu obnoví jeho původní platnost.

Z programu je možné zrušit záznam procedurou DbfDelete, obnovit platnost záznamu lze procedurou DbfRecall. To, zda je záznam určen ke zrušení zjistíme voláním funkce DbfDeteted. Pokud by jsme potřebovali zkontrolovat záznamy a obnovit jejich platnost v celé databázi, můžete tak učinit následujícím programem:

```
DbfGoTop;                {skok na začátek databáze}
WHILE not DbfEof         {dokud není konec databáze}
   IF DbfDeleted then    {pokud záznam označen ke zrušení}
      DbfRecall;         {obnovíme platnost záznamu}
   ENDIF
   DbfSkip(1);           {skok na další záznam}
ENDWHILE
```

Záznamy určené ke zrušení se z databáze definitivně odstraní voláním procedury DbfPack, která zruší najednou všechny označené záznamy v databázi. POZOR !!! Jde již o zrušení nevrané a označené záznamy budou definitivně z databáze odstraněny.

Někdy by mohlo být vhodnější, kdyby jsme nemuseli záznamy z databáze rušit, přesto by byly nepřístupné. To je samozřejmě také možné. Jednou z možností by bylo nastavit filtr podle požadované platnosti záznamů voláním procedury:

```
DbfSetFilter(".NOT.DELETED()");  {pouze platné záznamy}
DbfSetFilter("DELETED()");       {pouze neplatné záznamy}
```

Další a mnohem výhodnější a rychlejší je varianta, kdy se platnost záznamů nastaví přímo svým nastavením voláním procedury DbfSetDeleted, která určuje, zda budou dostupné záznamy označené ke zrušení:

```
DbfSetDeleted(False);    {dostupné pouze platné záznamy}
DbfSetDeleted(True);     {dostupné všechny záznamy}
```

Někdy by jste potřebovali zrušit najednou všechny záznamy z databáze. To by bylo možné

provést tak, že by jste postupně označili každý záznam databáze ke zrušení a následně je procedurou DbfPack zrušili. Možná je ale i výhodnější a rychlejší varianta, při které jsou voláním procedury DbfZap zrušeny najednou všechny záznamy databáze bez ohledu na to, zda jsou označeny ke zrušení. VAROVÁNÍ !!! Záznamy již nebude možné žádným způsobem obnovit !

44. lekce - Databázové oblasti

Systém KLONDAIK umí pracovat najednou až se třemi otevřenými databázemi. Každá databáze je přitom otevřena ve své samostatné datové oblasti.

Mezi datovými oblastmi je možné se přepínat v programu voláním procedury DbfSelect. Při práci přímo s databázovým oknem máte možnost použít výhodněji PopUp menu, které se zobrazí po stisku pravého tlačítka myši na ploše datové mřížky. Menu obsahuje buď čísla prázdných oblastí, případně uvedení jmen databází otevřených v jednotlivých datových oblastech. Požadovanou oblast zvolíte kliknutím myší.

Možnost práce s několika najednou otevřenými databázemi oceníte zejména při náročnějších požadavcích na zpracování dat. Data z jedné oblasti můžete například různě upravit a ukládat do databáze otevřené v jiné oblasti. Třetí datová oblast může přitom sloužit například jako číselník údajů, ceník pro výpočty a podobně.

45. lekce - Vytvoření nové databáze

V předchozích lekcích jsme si ukázali, jak se dá v systému KLONDAIK pracovat jednoduše s databázemi. Dosud jsme však používali pouze zkušební databáze, které se dodávají spolu se systémem. Proto se nyní seznámíme s možností vytvoření nové vlastní databáze a s možností změny struktury již existující databáze.

Ještě předtím, než začneme na počítači tvořit databázi by jsme si měli ujasnit, jaké údaje má naše databáze obsahovat a jakého jsou typu. Důsledný návrh nám následně umožní plné využití zadaných dat. Pokud si budete chtít například zadat vlastní databázi adres, máte mnoho možností jak definovat jednotlivé položky databáze. Například jméno může být obsaženo v jedné znakové položce. Pokud však vytvoříte samostatné položky pro jméno a příjmení, umožní vám to snadněji hledat nejen podle příjmení, ale i podle jména. Tím můžete například zjistit, kteří vaši zákazníci oslaví v zadaný den svůj svátek a budete jim moci zaslat blahopřání. Samostatnou položku můžete vyhradit například i pro titul. Pokud potom například "odfiltrujete" záznamy, které nemají uveden titul, získáte seznam kvalifikovaných odborníků. Obdobným způsobem by bylo možné probrat všechny požadované údaje. Pečlivý návrh databáze se vám rozhodně vyplatí, proto jej nepodceňujte.

Dalším krokem je stanovit si, jakého typu mají být jednotlivé položky. To se může stát někomu podivné, ale například poštovní směrovací číslo obsahuje pouze čísla, přesto je vhodné je definovat jako řetězec. Stejně tak jako ostatní údaje numerického vyjádření, se kterými se však neprovádí matematické výpočty (číslo účtu, střediska apod.). To mám následně umožní snadnější tvorbu indexů pro hledání v databází a jejich třídění.

Posledním krokem návrhu databáze je stanovení délek jednotlivých položek. U numerických položek současně stanovíme počet desetinných míst v uloženém čísle. Mějte při návrhu na paměti, že každý zbytečná znak se na disku v obsazeném prostoru násobí počtem záznamů.

Po provedení výše uvedeném návrhu databáze můžete zadat její strukturu do systému. Musíte k tomu aktivovat databázové okno a uzavřít případně databázi v první oblasti. Pouze tak se vám po stisku pravého tlačítka myši na ploše databázového okna zobrazí podmenu s možností výběru volby pro zadávání nové databáze. Pokud ji aktivujete, zobrazí se vám nové okno pro zadávání struktury databáze.

Do prázdné tabulky budete zadávat popis položek databáze. Nejprve stiskněte na tlačítko "Přidat", které umístí do tabulky popis nové položky. Uvedeno je náhodně zvolené jméno. Editací máte možnost změnit. Jméno položky databáze musí začínat na písmeno a smí mít délku maximálně deset znaků a nesmí obsahovat mezery. Jméno položky databáze musí být jednoznačné a nesmí se v databázi opakovat.

Dále musíte zadat typ položky. Jak již bylo uvedeno v předchozích lekcí, označuje se řetězcový typ položky písmenem "C", numerický typ položky písmenem "N", logický typ položky se označuje písmenem "L", datum se označuje písmenem "D" a poznámka libovolné délky písmenem "M". Poznámku nelze v aktuální verzi systému editovat, ani zobrazovat. Lze ji pouze vytisknout na sestavě.

Podle toho, jaký typ položky jste zadali, máte možnost definovat délku položky. Pouze u logické položky je pevně nastavena délka na jeden znak, datum zabírá vždy osm znaků a poznámka (memo položka) deset. Memo položka přitom obsahuje pouze adresu umístění poznámky v samostatném souboru, který má jméno shodné se jménem databáze a příponu FPT. Pouze u numerické položky, které může mít délku maximálně dvacet znaků můžete zadat navíc i počet desetinných míst, které se však spolu s desetinnou tečkou započítávají do celkové délky položky.

Výše uvedeným způsobem zadáte do tabulky popis všech položek databáze. Novou položku

přidáte kliknutím myší na tlačítko "Přidat". Položku databáze, na které je umístěn kurzor máte možnost zrušit stiskem myší na tlačítku s nápisem "Vymazat".

Po zadání a zkontrolování zápisu všech položek nové databáze musíte vytvořenou strukturu databáze uložit na disk počítače, což je možné provést kliknutím myší na tlačítko "Uložit do...". Zobrazí se běžný dialog Windows pro zadání jména nového databázového souboru, který musí mít vždy příponu *.DBF. V případě správného zadání je databáze vytvořena a je ihned zobrazena v databázovém okně s nastavením možnosti okamžité editace.

46. lekce - Změna struktury databáze

Podobným způsobem, jako je zadání struktury nové databáze je možné provést i její změnu. Tentokrát musíte mít databázi, jejíž strukturu budete chtít změnit otevřenou v první oblasti databázového okna. Po kliknutí pravým tlačítkem myši se zobrazí podmenu s přístupnou volbou umožňující změnu struktury databáze. Po jejím výběru se zobrazí již známá tabulka s možností změny položek databáze. Máte přitom možnost již známým způsobem přidávat nové položky databáze, případně zrušit již existující položky. Máte také možnost provést změnu parametrů jednotlivých položek.

Provedené změny máte možnost uložit dvěma způsoby. Pokud kliklnete myší na tlačítko "Uložit do...", budete mít možnost zadat jméno databáze, do které se data uloží. Původní databáze zůstane přitom zachována beze změny. Pokud zadáte jméno již existující databáze, budete dotázáni na její přepsání. V případě souhlasu budou data zvolené databáze nahrazena novou databází.

Pokud budete chtít uložit modifikovanou strukturu databáze do stejného souboru, stiskněte myší tlačítko "Uložit". Dojde ke změně struktury databáze s přihráním původního obsahu. Původní databáze zůstane přitom zachována v souboru TEMP_BAK.DBF. Pokud obsahuje databáze memopoložky, je jejich obsah zachován v záložním souboru TEMP_BAK.FPT. Musíte si uvědomit, že při další provedené změně struktury jsou uvedené soubory nahrazeny vždy novým obsahem.

V obou výše uvedených případech dojde ke změně struktury dat podle nového zadání databázové struktury. Nově přidané položky jsou samozřejmě prázdné, Položky, u kterých jste změnili délku mohou přijít o část svého obsahu. Neprovádějte změnu typu položek, protože by nebylo možné převzít do nové databáze původní data. Pokud tak budete muset učinit, nadefinujte ve struktuře databáze novou položku požadovaného typu, proveďte změnu struktury a následně sestavte program, kterým převedete data do nové položky s příslušnou konverzí. Další modifikací struktury databáze můžete původní položku zrušit.

47. lekce - Editace memo položek databází

Jak jsme se již seznámili v předchozích lekcích, je možné ukládat v databázích údaje různého typu. Běžné jsou údaje typu číslo, řetězec, datum a logická hodnota, jejichž délka je vždy definována ve struktuře databáze. Pokud budeme potřebovat ukládat do databáze údaj, jehož délka je proměnná, můžeme použít k jejich uložení tzv. Memo položky.

Memo položka je ve struktuře databáze definována vždy o délce 10 bajtů a má uveden typ "M". Skutečná délka memo položky však může být proměnná, protože se do databáze ukládá ve skutečnosti pouze adresa uložení memo položky v samostatném souboru, který tvoří nedílnou součást databáze. Uvedený soubor má vždy jméno shodné se jménem databáze, jeho přípona je však vždy *.FPT. Uvedený soubor se otevírá automaticky při otevírání databáze a nemusíte s ním provádět žádné akce. Pokud je ve struktuře databáze definována memo položka a příslušný soubor, který je má obsahovat neexistuje, bude systémem nahlášena chyba a databáze nebude otevřena.

Memo položky nejsou přístupné přímo z programu. Máte k mim přístup pouze z okna s databází. Pokud obsahuje databáze memo položku, je tato v okně databáze indikována v celém sloupci textem "(Memo)". Pokud v uvedeném sloupci stisknete klávesu Enter, zobrazí se obsah memo položky. Forma zobrazení závisí na tom, zda máte nastavenou možnost editace záznamů databáze. Pokud ne, zobrazí se v novém okně pouze obsah memo položky. Zpět se přepnete stiskem klávesy Esc. Pokud máte nastavenu možnost editace databáze budou v okně zobrazeny navíc i tlačítka OK a Zrušit pro ukončení editace memo položky. Možnost editace databáze se nastaví přes pomocné menu, které se zobrazí po stisku pravého tlačítka myši na ploše databázového okna.

V memo položce máte možnost se pohybovat pomocí kurzorových kláves. Lze také běžným způsobem vybrat blok textu a umístit jej do schránky Windows. To vše bez ohledu na to, zda máte povolenou editaci záznamů. Text obsahu memo položky se automaticky zarovnává ke krajům. Změnou velikosti okna máte proto možnost ovlivnit zobrazení textu.

Formulář se zobrazeným obsahem memo položky je zobrazen modálně, to znamená, že se nelze přepnout do jiného okna. Ukončení zobrazení (editace) se provede stiskem klávesy Esc. Pokud byl obsah memo položky změněn, jste dotázáni, zda se mají provedené změny uložit. Pokud máte nastavenou možnost editace, máte možnost ji ukončit navíc i tlačítkem OK, kdy se provedené změny uloží do databáze, případně tlačítkem Zrušit, kdy jsou změny anulovány.

UPOZORNĚNÍ:

Pokud budete pracovat s databází, která byla vytvořena v jiném systému a memo položka bude delší než 32 Kb, nebude možné tuto memo položku zobrazit.

48. lekce - Tisk sestav z databází (reporty)

Databází je možné prohlížet dvěma různými způsoby. V předchozích lekcích jsme se již seznámili s možností zobrazování databáze v databázovém okně, kterému se také v databázových systémech říká BROWS. Nyní se seznámíme s další možností a to je zobrazení sestavy formou sestavy (reportu). Sestava je zobrazení databáze ve formě tiskového výstupu. Sestavu je možné kromě tisku na tiskárně také zobrazit na obrazovce.

Před definicí sestavy si nejprve musíte otevřít požadovanou databázi, ze které budete chtít sestavu tisknout. Dále můžete omezit rozsah dat nastavením filtru. Pokud budete chtít řadit data v sestavě podle indexu, musíte jej také nastavit.

Tvorbu sestavy zahájíte kliknutím myší na ikonu s tiskárnou v databázovém okně. Standardně se nabízí sestava se jménem databáze a příponou *.PTS. Název můžete změnit. Pokud již předloha zadané sestavy existuje, bude zobrazen její návrh k možným dalším úpravám. V opačném případě vám nabídnuto její vytvoření.

Při vytváření nové sestavy jsou nabídnuty všechny položky databáze k jejich výběru zařazení do sestavy. V levé části dialogového okna inicializace sestavy jsou nabídnuty položky databáze, v pravé části jsou umístěny položky, které budou umístěny v sestavě. Uprostřed se nachází tlačítka, kterými můžete přesouvat položky databáze. Dále obsahuje dialogové okno inicializace sestavy výběr způsobu tisku. Buď jako běžná sestava, kdy jsou položky databáze umístěny vedle sebe, nebo v řádcích pod sebou. Po ukončení inicializačního dialogu je vygenerována předloha sestavy dle zadaného nastavení.

Předloha sestavy je zobrazena v samostatném okně, které slouží nejen k její editaci, ale také k zobrazení finálního vzhledu sestavy. V horní části okna se nachází panel nástrojů pro editaci předlohy sestavy, která se nachází v dolní části okna. Jednotlivé prvky sestavy máte možnost přesouvat po ploše předlohy sestavy, nastavovat parametry tisku a podobně. Je také možné doplnit sestavu dalšími prvky. Buď grafickými (obrázek) nebo textovými (nadpisy, datum tisku, číslo strany apod.). Předloha sestavy je rozdělena na několik samostatných oddílů. Začátek sestavy je tisknut pouze jednou na začátku sestavy. Hlava stránky se tiskne na začátku každé stránky a obsahuje nejčastěji popis sloupců sestavy. Tělo sestavy obsahuje vlastní data sestavy. Pata stránky se tiskne na konci stránky. Podrobnosti o použití jednotlivých nástrojů editace sestavy naleznete v samostatné nápovědě, kterou máte možnost vyvolat kdekoliv z editace předlohy sestavy stiskem klávesy F1.

Sestavu je možné vytisknou i z programu voláním procedury DbfReport. Volbou parametrů procedury je možné zadat rozsah stránek pro tisk a to, zda má být sestava ihned tisknuta na tiskárně, případně zda se má nejprve zobrazit její previev na obrazovce.

49. lekce - Co je to tabulka, k čemu se používá

Tabulka je skupina buněk uspořádaných v řádcích a sloupcích ve tvaru mřížky. Jednotlivé buňky tabulky mohou obsahovat buď číselné údaje, texty nebo vzorce pro výpočet obsahu buňky. Změna obsahu buňky může vyvolat opakovaný výpočet obsahu jiných buněk na základě definovaných vztahů mezi buňkami. Tabulka je proto velmi vhodná na výpočet soustavy hodnot. Jako tabulku si můžeme například představit matici čísel, která se má sečíst ve sloupcích i řadách. Výhodou tabulky přitom je, že změnou libovolného čísla mohou být ihned přepočítány ostatní hodnoty a je možné takto analyzovat situaci změnou hodnot.

Příkladem jednoduché tabulky může být například výpočet ročního zisku z údajů o příjmech a výdajích za kvartál. V prvním sloupci je uvedeno období, ve druhém příjmy a ve třetím sloupci jsou uvedeny výdaje. V posledním sloupci je potom proveden výpočet zisku za kvartál (příjem-výdej). Čísla jsou ve sloupcích sečtena a na posledním řádku je uveden výpočet příjmů a výdajů za celý rok. Poslední buňka vpravo dole obsahuje konečný výpočet ročního zisku. Tabulka může mít na papíře například následující podobu:

	příjem	výdej	zisk
1.kvartál	100	60	40
2.kvartál	120	40	80
3.kvartál	80	90	-10
4.kvartál	180	110	70
součet	480	300	180

Při změně libovolného číselného údaje musíte tabulku přepočítat. Pokud je tabulka jednoduchá, jako v našem uvedeném případě, není to i bez použití kalkulačky těžké. V mnoha skutečných případech však bývají tabulky mnohem složitější. Proto je výhodné použít pro výpočet tabulek počítač. To vám umožní navíc vyzkoušet i různé varianty výpočtů typu co by se stalo, kdyby... Můžete změnit číselné hodnoty v buňkách a ihned máte možnost vyhodnotit změny, které se zobrazí v přepočítané tabulce.

Složitější tabulky se mohou použít například pro výpočty splátek, úvěrů, odpisu majetku, ale například i daňového přiznání. Počáteční zadání tabulky může být v takovém případě složitější a je nutné uvést všechny vazby mezi čísly obsaženými v tabulce. Výhodou potom je rychlost, a neomylnost, se kterou potom dokáže tabulka reagovat na změny některých hodnot.

50. lekce - Zadávání hodnot, editace tabulky

Jak již bylo uvedeno, skládá se tabulka z buněk, do kterých lze ukládat čísla, texty nebo vzorce. Nová tabulka je prázdná a neobsahuje žádné údaje. Aby bylo možné do některé buňku zapsat údaje, musí se tabulka nejprve přepnout do editačního módu. Přepínání do editačního režimu se provádí kliknutím na příslušnou ikonu v horní liště okna s tabulkou. Zobrazí se přitom editační řádek pro zadávání údajů do tabulky a označení aktivní buňky. Aktivní buňka je přitom ta, která je označena zvýrazněným rámečkem. Zapíšete-li do editačního řádku tabulku libovolný text, zobrazí se současně i v aktivní buňce. Ukončení zápisu do buňky musíte provést vždy stiskem klávesy Enter.

Chcete-li zapisovat do jiné buňky, můžete měnit aktivní buňku pomocí kurzorových kláves. Obsah buňky můžete i jednoduše změnit. Začnete-li po změně aktivní buňky psát text, bude nejprve původní obsah buňky zrušen a zápis se bude provádět do prázdné buňky. Jestliže budete chtít původní obsah opravit, stiskněte klávesu F2, čímž se přesune obsah buňky do editačního řádku a vy máte možnost jej změnit.

Obdobným způsobem, jako se zapisuje do tabulky text můžete zapsat i číselné hodnoty. Nesmíte přitom zapsat nenumerický znak, protože v takovém případě by se obsah buňky posuzovat jako text a nebylo by možné s ním samozřejmě provádět žádné výpočty.

Pokud zapisujete do buňky text, který je delší, než šířka tabulky, může být text zobrazen i v oblasti sousedních buněk. Pouze však v případě, že jsou sousední buňky prázdné. Pokud tak není, bude text uříznut a zobrazí se pouze jeho část.

Každý řádek a sloupec buněk má své označení. Řádky jsou číslovány a sloupce značeny písmeny podle abecedy. Označení řádků a sloupců je přitom viditelné v levém a horním okraji tabulky. Každou buňku tabulky lze tedy určit jednoznačně označením sloupce a řádky. Například buňka ve třetím řádku a druhém sloupci má označení B3. Uvádí se přitom vždy nejprve písmeno a potom číslo (bez mezery). Pokud máte přepnutou tabulku do editačního režimu, zobrazuje se označení aktivní buňky na tlačítku umístěném vlevo od editačního řádku. Pokud kliknete na tlačítko myší, zobrazí se dialog umožňující zadání označení buňky, která se má aktivovat.

Zatím jsme si ukázali, jak lze do tabulky zapsat číselné a textové údaje. Žádné výsledky se ale zatím nepočítají, protože jsme nezadali vzorce pro výpočet hodnot. To si ukážeme až v následující lekci.

51. lekce - Zadání vzorců pro výpočet tabulky

Tabulka by kromě číselných hodnot měla obsahovat i definici vzorců pro výpočet výsledků. Vzorce jsou přitom zadány vždy do buňky, ve které má být zobrazen výsledek výpočtu. Vzorec přitom obsahuje zadání výpočtu výsledné hodnoty.

Použití vzorců si pro jednoduchost ukážeme na sečtení dvou čísel, která jsou například uložena v buňkách B2 a B3. Výsledek součtu má být přitom uložen v buňce B4. Aktivujeme proto uvedenou buňku B4 a do editačního řádku zapíšeme vzorec pro výpočet výsledné hodnoty:

=B2+B3

Všimněte si, že vzorec musí začínat vždy rovnítkem ! Ukončíme-li zápis vzorce stiskem klávesy Enter, dojde k výpočtu výsledku a jeho zobrazení. Změníme-li vstupní hodnoty, dojde ihned k přepočtu výsledku. Obdobným způsobem je možné zadat i vzorce pro výpočet dalších výsledků.

V příkladu jsme čísla v buňkách sečetli. Použili jsme proto operátor "+". Obdobně ale můžeme použít i další operátory a závorky. Pro výpočet mocniny použijeme znak "^". Vzorce mohou mít proto tvar například =C2*3, =B1/B2-C3 nebo při složitějším zápisu =A3*(B6+B7), =B5^2. Při výpočtu se dodržuje zásada, že násobení a dělení má přednost před sčítáním a odečítáním. Pokud se mají proto sečíst dvě čísla a jejich součet vynásobit jiným číslem, je nutné použít ve vzorci závorky.

Pokud jsou vzorce jednoduché, není problém zadat z klávesnice do editačního okna požadovaný výpočet. Při použití složitějších výpočtů je ale výhodnější použít místo vypisování označení buněk kliknutí levým tlačítkem myši na požadované buňce, jejíž označení se ihned doplní do editovaného vzorce na místě kurzorou v editačním řádku. Můžete tak proto v našem případě při zápisu vzorce napsat úvodní rovnítko, kliknout na buňku B2 (ukážeme tím na odkaz, který se má do vzorce doplnit). Dále stiskneme na klávesnici plus a klikneme myší na buňku B3 a vzorec potvrdíme stiskem klávesy Enter. Takto je možné zadat vzorec mnohem jednodušeji a hlavně bez chyby při zápise odkazů.

Kromě běžných matematických vzorců je možné používat i mnoho dalších funkcí. POZOR ! Jedná se o funkce, které jsou dostupné pouze při výpočtech tabulek. Nejedná se o funkce systému KLONDAIK používané v ostatních částech systému (například pro práci s řetězci).

Použití funkcí tabulky si ukážeme na zabudované funkci ROUND, která se používá jak již jistě správně tušíte pro zaokrouhlování čísel. Každá funkce je ve vzorci následována závorkami, do kterých se uvádějí parametry pro výpočet. Funkce ROUND má přitom dva parametry. První parametr udává číslo, které se má zaokrouhlit a ve druhém parametru je uvedeno, na kolik desetinných míst se má číslo zaokrouhlit. Parametry se oddělují čárkou. Pokud budeme chtít například zaokrouhlit číslo uložené v buňce B2 na dvě desetinná místa, zadáme vzorec =ROUND(B2,2).

Při zápisu funkcí lze používat libovolné matematické výpočty. Parametry funkce mohou být odkazy na další buňky a podobně. Je proto možné psát například vzorce =A3*ROUND(A1,0)-A2 nebo =ROUND(A1+A2,2).

Další velmi výhodnou vlastností pro zadávání vzorců je možnost zadávání označení buněk v bloku. Pokud by jsme potřebovali sečíst několik buněk ležících pod sebou, mohli by jsme použít vzorec =A1+A2+A3+A4+A5. Výhodnější je ale použít funkce SUM a buňky pro výpočet zadat blokem. V takovém případě lze zadat vzorec vztahem =SUM(A1:A5). Uvedená dvojtečka přitom neznamená dělení (pro které se ostatně používá znak lomítko), ale ohraničení bloku, se kterým se funkce použije. Začátek bloku udává buňka uvedená před dvojtečkou, buňka za dvojtečkou potom uvádí konec bloku. Blokem přitom může být označen najednou rozsah několika řádků a sloupců (například C2:F6).

Pokud budeme chtít použít při psaní vzorců s bloky pro přenos označení buněk myši, zapíšeme

do editačního řádku rovnítko, levou závorku, klikneme myší na počáteční buňku bloku a při stisknutém tlačítku myši se přesuneme na konečnou buňku bloku. Současně s tím se opět přenese označení bloku do editačního řádku.

Uvedeným způsobem je možné čísla v tabulce nejen sčítat, ale při použití jiných funkcí lze například zjistit průměr čísel v bloku, maximální a minimální hodnotu a podobně. S dalšími užitečnými vzorci se seznámíme v následující lekci.

52. lekce - Funkce pro výpočty v tabulce

Při zápisu vzorců, které jsou ukládány do buněk tabulky lze použít mnoho definovaných funkcí. Funkce přitom slouží například pro matematické výpočty, práci s řetězci, konverzní funkce a podobně. Pozor, uvedené funkce (ukládané do buněk tabulky) nejsou slučitelné s funkcemi, které jsou definovány pro systém KLONDAIK (používané v programu).

Jak již bylo uvedeno, je tabulka včetně vzorců slučitelná s tabulkami systému EXCEL verze 4 (anglická varianta!). Podrobnosti o možnostech používání vzorců proto naleznete v příslušné literatuře k uvedenému systému. Seznam funkcí s přehledem používaných parametrů naleznete v nápovědě k systému KLONDAIK. Následuje pouze velmi zkrácený přehled nejzákladnějších možných funkcí:

ABS	- vrací absolutní hodnotu čísla
DATE	- vrací pořadové číslo zadaného dne
DAY	- převede pořadové číslo dne na den v měsíci
DATEVALUE	- převede datum z řetězce na pořadové číslo
FACT	- vrátí faktoriál čísla
INT	- zaokrouhlí číslo na nejbližší menší celé číslo
LEN	- vrátí počet znaků textového řetězce
MID	- vrátí zadaný počet znaků z řetězce od zadané pozice
MOD	- vrátí zbytek po dělení čísla
MONTH	- převede pořadové číslo dne na měsíc
RAND	- vrátí náhodné číslo z intervalu 0 až 1
REPLACE	- nahradí znaky v textu
ROUND	- zaokrouhluje číslo na zadaný počet číslic
SEARCH	- hledá v textu zadaný podřetězec
SUM	- sečte zadané argumenty
SUBSTITUE	- nahradí v textu zadaný řetězec jiným
TIME	- vrací pořadové číslo zadaného času
TODAY	- vrátí pořadové číslo na den v týdnu
TRIM	- odstraní nadbytečné mezery v textu
TRUNC	- odřízne desetinnou část čísla
VALUE	- převádí textový argument na číslo
WEEKDAY	- převádí pořadové číslo na den v týdnu
YEAR	- převádí pořadové číslo na rok

53. lekce - Přesun a mazání bloků v tabulce

V předchozích kapitolách jsme se již seznámili, jak je možné používat myš pro převod označení buněk a bloků do editačního řádku při zadávání vzorců. Nyní se seznámíme s dalšími možnostmi použití myši při tvorbě a editaci tabulek.

Někdy je nutné přenést obsah tabulky na jiné místo. To je samozřejmě možné provést novým zápisem na nové pozici. Výhodnější je ale použití myši. Aktivní buňka je vždy ohraničena rámečkem. Pokud najedeme myší na rámeček, změní se kurzor na šipku. Nyní stiskneme levé tlačítko myši a zatímco jej držíte, přesunete kurzor na buňku, kam chcete obsah přesunout. Poté tlačítko myši pustíte. Obsah buňky na původním místě zmizí a přesune se na pozici nové buňky. Pokud při této akci podržíte klávesu CTRL, nedojde k přesunu obsahu, ale k jeho kopii. Původní obsah buňky zůstane zachován.

Pokud obsahují kopírované buňky vzorce, dojde jejich kopírováním k přepočtu odkazů, změní se odkazy na buňky. Platí přitom, že o kolik buněk se vzorec posune, o tolik se posunou i odkazy ve vzorci. Pokud máme například v buňce B2 uvedeny příjmy a v buňce C2 výdaje, zadáme do buňky D2 zisk vzorcem =B2-C2. Pokud obsahují uvedené buňky hodnoty pro jeden rok a chtěli by jste zadat obdobné buňky pro další rok, stačí označit buňky B2 až D2 do bloku a ten zkopírovat o řádek níže. Všimněte si potom, že v buňce D3 bude vzorec =B3-C3. Došlo tedy ke změně odkazů vzorce a není již nutné vzorec zadávat znovu, ani opravovat.

Pokud potřebujeme, aby zůstal odkaz v buňce beze změny, musíme použít absolutní, tedy pevné odkazy na konkrétní buňky. Pokud například umístíte do některé buňky konstantní hodnotu, která se nemění (například procento DPH nebo kurs měny), můžete se na uvedenou buňku odkazovat z celé tabulky. Pokud by jste však použili blokový přesun, došlo by současně ke změně vzorce s odkazem na pevnou pozici buňky. Je proto nutné použít absolutní označení pozice, které se nepřepočítává. Při absolutním označení buňky se umístí před označení sloupce i řádky znak $. Ve vzorci se proto uvede například výpočet =B2+B3. Při kopírování přitom nedojde ke změně odkazu B2, ale pouze u odkazu B3.

Až dosud jsme používali označování bloků uvnitř tabulky. Pokud však budete potřebovat označit najednou jako blok celý řádek, případně sloupec, můžete tak učinit kliknutím myši na záhlaví sloupce nebo řádky. Tím dojde k označení buď celého řádku, případně sloupce. Pokud potřebujeme označit najednou několik řádků, klikneme myší na první požadovaný řádek, přejedeme na poslední řádek a myš uvolníme. Takto označíme souvislý blok několika řádků. Obdobně máme možnost označit několik sloupců.

Pokud potřebujete vložit do tabulky nový řádek nebo sloupec, označíte řádek tabulky a stisknete klávesu CTRL a klávesu plus na numerické klávesnici. Do tabulky se vloží nový prázdný řádek a následující řádky se posunou dolů. Pokud ale stisknete klávesy CTRL a klávesu mínus, dojde k výmazu označené řádky. Podobný postup lze použít i v případě, pokud chcete přidat do tabulky sloupec, případně označený sloupec vymazat.

V předchozích lekcích jsme se naučili tabulku editovat, zadávat vzorce pro výpočet a provádět základní úpravy v tabulce. V následujících lekcích si ukážeme, jak tabulku uložit do souboru a jak upravit vzhled tabulky. Nakonec se ještě samozřejmě naučíme, jak přistupovat k tabulce přímo z programu.

54. lekce - Inicializace, uložení a načtení tabulky

Založit novou tabulku můžete několika způsoby. Nejjednodušší je kliknout na příslušnou ikonu v horní liště okna s tabulkou. Tím dojde nejen k výmazu obsahu tabulky, ale i vzorců zadaných v tabulce. Pokud by jste chtěli založit novou tabulku s požadovaným počtem řádek a sloupců, můžete použít volání procedury SpreadInit z programu, případně příkazového okna. Jako parametry procedury se přitom uvádí počet řádků a sloupců požadované tabulky.

Vytvořenou tabulku budete chtít samozřejmě uložit na disk do souboru k dalšímu možnému použití. K tomu použijte příslušnou ikonu umístěnou v horní liště okna s tabulkou. Tabulku můžete uložit ve dvou různých formátech. Buď ve vlastním optimalizovaném formátu *.VTS, případně ve formátu známého programu EXCEL ve verzi 4. Formát tabulky *.VTS používá i několik dalších systémů - například Grafiline Calc. Typ ukládané tabulky se určuje uvedením přípony souboru. Tabulku můžete také uložit do souboru použitím procedury SpreadSave. Mezi oběma formáty můžete provádět jednoduše konverzi tak, že načtete tabulku s původním typem a pro její uložení použijete typ druhý. Původní tabulka zůstane přitom zachována beze změny.

Pro načtení tabulky do systému můžete použít samozřejmě opět buď ikonu v horní liště okna s tabulkou, případně volání procedury SpreadLoad z programu nebo příkazového okna.

55. lekce - Grafické úpravy tabulky

Vytvořenou tabulku máte možnost dále graficky upravit. Můžete například nastavit velikost řádků a sloupců, velikost fontu, barvy, zarovnávání, okraje, čáry a podobně.

Nejjednodušší je nastavení velikosti řádků a sloupců, to znamená změnit jejich vzájemné rozestupy. Toho dosáhnete tak, že uchopíte myší dělící čáru mezi řádky (sloupci) tabulky v prostory jejich záhlaví a táhnutím změníte velikost řádky.

Pro další úpravy si již budete muset vždy nejprve označit myší oblast, pro kterou se mají požadované úpravy provést. Pokud by jste tak neučinili, byly by provedeny změny pouze u aktivní buňky tabulky. Některé změny v uspořádání tabulky je možné provést kliknutím na ikonu umístěnou v horní liště okna s tabulkou. Ikony ve dvou skupinách obsahují úpravy zarovnávání a zobrazení číselných hodnot. V první skulině jsou tři ikony pro zarovnávání obsahu buněk vlevo, doprostřed a vpravo. Další skupina ikon umožní změnit číselné hodnoty včetně měny, v procentech nebo v exponenciálním tvaru.

Další změny v tabulce máte možnost nastavit z PopUp menu, které se zobrazí po stisku pravého tlačítka na ploše tabulky. Po výběru položky menu se zobrazí dialogový box umožňující nastavení zvolených atributů tabulky.

Volbou 'Font' z PopUp menu máte možnost nastavit ve vybrané oblasti požadovaný font včetně barvy zobrazení. Pokud budete chtít nastavit čáry ohraničující zvolenou oblast, zvolte volbu 'Okraje', které umožňují nastavit nejen typ čáry, ale opět i jejich barvu. Volbou 'Plocha' máte potom možnost nastavit plochu vybrané oblasti, to je barvu a podklad.

Uvedené změny není možné nastavit z programu. Pokud by jste potřebovali používat v programu graficky upravenou tabulku, musíte si vytvořit nejprve její vzor, který uložíte do souboru. Potom již můžete připravený vzor tabulky využít ve svém programu.

56. lekce - Tisk tabulek

Vytvořenou tabulku je možné vytisknout na tiskárně. Nejprve je ale vhodné provést nastavení způsobu tisku. Příslušný dialog nastavení je umístěn v PopUp menu, které se zobrazí po stisku pravého tlačítka myši na ploše tabulky. Vyberte přitom volbu 'Stránka'.

V nastavení stránky pro tisk máte možnost zadat text záhlaví a ukončení stránky. Do textu je možné přenést některé systémové informace. K tomu slouží znak &, za který se uvede symbol příslušné informace dle následující specifikace:

&D - aktuální datum
&T - aktuální čas
&F - jméno tabulky (souboru)
&P - číslo stránky
&N - celkový počet stránek

Pokud chcete například vytisknout jméno souboru s tabulkou, zadáte v definici záhlaví nebo paty stránky text 'Tisk tabulky ze souboru &F'. Obdobně můžete zadat například datum tisku a číslo strany uvedením parametru 'Strana číslo &P, datum tisku:&D'.

V další části dialogu nastavení stránky tisku tabulky máte možnost nastavit velikost okrajů stránky. Míry se zadávají v palcích (palec = 25.54 mm).

V pravé horní části dialogového okna se zadává způsob tisku rozsáhlých tabulek, které není možné vytisknout na jednom listu. Udává se, zda se má tisknout odshora dolů, nebo zprava doleva. V další části dialogu máte možnost nastavit umístění tabulky. Tisknout se bude buď od definovaných okrajů, případně je možné zadat horizontální a vertikální centrování tabulky.

V pravé dolní části dialogového okna pro nastavení tisku tabulek lze nastavit, zda se mají tisknou čáru mřížky tabulky, zda tisknout tabulku černobíle (pozor na možnosti a nastavení tiskárny) a zda tisknout záhlaví řádků a sloupců.

POZOR !

Před vlastním tiskem tabulky musíte nejprve označit oblast, která se má vytisknout. Pokud by jste tak neučinili, vytiskla by se pouze aktivní buňka. Označit celou tabulku lze přitom jednoduše kliknutím na levou horní část záhlaví tabulky. Tisk tabulky se zahájí kliknutím myší na příslušnou ikonu umístěnou v horní liště okna s tabulkou.

57. lekce - Procedury a funkce pro práci s tabulkou

Systém obsahuje několik zabudovaných procedur a funkcí, které je možné použít k přístupu k tabulce. V předchozích lekcích jsme se naučili tabulku inicializovat procedurou SpreadInit, načíst tabulku ze souboru procedurou SpreadLoad a uložit tabulku do souboru procedurou SpreadSave. Zrušení obsahu tabulky beze změny její velikosti se provede procedurou SpreadClear. Okno s tabulkou ukryjete procedurou SpreadHide. Původní velikost tabulky obnovíte procedurou SpreadShow. Změnu velikosti okna s tabulkou provedete procedurou SpreadForm.

Pokud budete chtít ukládat do tabulky nové hodnoty, případně číst hodnoty z tabulky, budete muset nejprve zadat požadovanou buňku tabulky. Sloupec tabulky se zadává procedurou SpreadCol, řádek tabulky se zadává procedurou SpreadRow. Tím se stane zadaná buňka aktivní. Číselnou hodnotu uložíte do tabulky procedurou SpreadNumber. Text můžete do buňky uložit procedurou SpreadText. Vzorec pro výpočet obsahu aktivní buňky zadáte procedurou SpreadFormula. Pokud zadáváte vzorec pro výpočet obsahu buňky z programu, nesmíte uvádět počáteční rovnítko ve vzorci!

Data lze do tabulky z programu nejen ukládat, ale je možné také načíst výsledné hodnoty z libovolné buňky. Požadovanou buňku musíte nejprve aktivovat procedurami SpreadRow a SpreadCol. Číselnou hodnotu přečtete z tabulky funkcí SpreadGetNumber. Text můžete přečíst z tabulky funkcí SpreadGetText. Následuje příklad využití vytvoření a naplnění tabulky přímo z programu.

```
// kompletní program je uložen v souboru ZISK.IPS
PROCEDURE main
   SpreadInit(17,4)
   SpreadRow( 1); SpreadCol(2); SpreadText("ZISK FIRMY");
   SpreadRow( 2); SpreadCol(1); SpreadText("Měsíc")
   SpreadCol( 2); SpreadText("Příjem");
   SpreadCol( 3); SpreadText("Výdej");
   SpreadCol( 4); SpreadText("Zisk");
   SpreadRow( 3); SpreadCol(1);
   SpreadText("====================================");
   SpreadRow( 4); SpreadCol(1); SpreadText("Leden");
   SpreadCol( 4); SpreadFormula("SUM(B4-C4)");
   SpreadRow( 5); SpreadCol(1); SpreadText("Únor");
   SpreadCol( 4); SpreadFormula("SUM(B5-C5)");
   SpreadRow( 6); SpreadCol(1); SpreadText("Březen");
   SpreadCol( 4); SpreadFormula("SUM(B6-C6)");
// .. obdobně se zadají měsíce duben až prosinec
// .. a dokončíme konec tabulky se součty
   SpreadRow(16); SpreadCol(1);
   SpreadText("====================================");
   SpreadRow(17); SpreadCol(1); SpreadText("Součet");
   SpreadCol( 2); SpreadFormula("SUM(B4:B15)");
   SpreadCol( 3); SpreadFormula("SUM(C4:C15)");
   SpreadCol( 4); SpreadFormula("SUM(D4:D15)");
// naplníme tabulku náhodnými čísly
   FOR X := 4 TO 15;
       SpreadRow(X);
       SpreadCol(2); SpreadNumber(Random(100));
       SpreadCol(3); SpreadNumber(Random(100));
   ENDFOR
```

```
// a zobrazíme výslednou hodnotu převzatou z tabulky
   MessageBox("Roční zisk firmy je: "
              +IntTostr(SpreadGetNumber),"Výsledek",0);
ENDPROC
```

Další procedury a funkce pro práci s tabulkami je možné použít pro úpravu vzhledu tabulky. Pokud nebudete chtít zobrazovat mřížku tabulky, použijte volání procedury SpreadShowLines. Zobrazení záhlaví sloupců a řádek můžete změnit procedurou SpreadShowColHead a SpreadShowRowHead.

Pokud je tabulka obsáhlá a provádíte rozsáhlé změny hodnot, mohlo by být někdy výhodné dočasně vypnout automatický přepočet tabulky procedurou SpreadAutoRecalc. Přepočítat tabulku je potom možné voláním procedury SpreadRecalc.

V předchozích lekcích jsme se seznámili s tím, jak lze snadno propočítat množství dat umístěných v tabulce. Pokud by jste potřebovali data vhodně graficky zobrazit, máte možnost použít zobrazení dat ve formě grafů, se kterými se seznámíme již v následující lekci.

58. lekce - Co je to graf, jak jej používat

V předchozích lekcích jsme se seznámili s tím, jak je možné provést výpočty hodnot v tabulkách. Při vhodném navržení tabulky by bylo možné přitom získat představu o vývoji hodnot. Dokonalejší vyhodnocení průběhu hodnot vám ale poskytnou pouze grafy, se kterými se seznámíme v několika následujících lekcích.

Jak již vyplývá z jejich názvy, používají se grafy ke grafickému vyhodnocování průběhu veličin. Většinou nás nezajímají přesné hodnoty, ale chceme porovnat vzájemný poměr hodnot, jejich nárůst a klesání. Poslouží nám tak například k rychlému posouzení příjmů a výdajů, ale také například k vyhodnocení závislost vzdálenosti na čase a podobně.

V grafu lze sledovat vzájemný vliv různých veličin. Lze například zobrazit závislost vzdálenosti, kterou urazí cyklista za určitý čas. Na grafu by bylo možné například sledovat vliv únavy, protože s rostoucím časem není cyklista schopen podávat stále maximální výkon. Číselně je možné zobrazit takový graf dvojicí čísel, kdy se na úsečce zobrazující jednu veličinu (čas) zaznamenává druhá veličina (vzdálenost). Pokud přitom zobrazíme druhou veličinu úsečkou kolmou k první veličině, získáme nejjednodušší graf.

K výše uvedenému grafu můžete přidat i další veličiny, kterými mohou být v tomto případě například naměřené hodnoty dalších cyklistů. Díky tomu by jsme mohli sledovat to, jak působí na různé cyklisty únava v závislosti na době jízdy. Takový graf se číselně zobrazuje jako tabulka.

Dle výše uvedeného popisu vyplývá, že graf se skládá z číselné části (hodnoty grafu) a grafické části (zobrazení průběhu hodnot). Číselná část grafu slouží pouze jako prostředek pro zadání hodnot. Hodnoty grafu mohou být získány výpočtem, případně zadáním hodnot. Grafická část naopak slouží k vyhodnocení a posouzení průběhu hodnot. Je přitom možné změnit nastavení zobrazení typu grafu a získat zobrazení grafu z několika různých pohledů.

59. lekce - Zadání hodnot grafu, změna zobrazení

Před prvním použitím grafu musíme systému nejprve zadat, jak bude graf veliký. To provedeme jeho inicializací. Inicializací grafu se ztratí jeho případné předešlé hodnoty a je proto vhodné si zvážit důkladně předem požadovanou velikost grafu. Inicializace grafu se provede voláním procedury ChartInit z programu, případně přímo z příkazového okna. Jako parametry procedury se uvádí počet zadávaných hodnot na ose x (směr do hloubky třírozměrného grafu) a osa y (vodorovná osa grafu). Pokud budete zadávat dvourozměrný graf, zadáte rozměr x grafu roven nule, nebo jedné.

Od verze 2.01 systému KLONDAIK je možné inicializovat graf i kliknutím na inicializační ikonu umístěnou v horní liště okna s grafy. Zobrazí se dialogový box, ve kterém zadáte požadované rozměry grafu pro inicializaci.

Dle zadané inicializace grafu se zobrazí prázdný graf. Je již sice možné přepínat typ grafu, ale vzhledem k tomu, že neobsahuje žádné hodnoty, nezobrazuje graf žádné veličiny. Hodnoty do grafu je možné zadat buď z programu výpočtem, případně přímým zadáním hodnot. Pro přímé zadávání hodnot klikněte myší na editační ikonu umístěnou v horní liště okna grafu (ikona s čísly). Zobrazí se tabulka jejíž počet řádků a sloupců odpovídá zadané inicializaci grafu. Pozor, tato tabulka nemá nic společného s tabulkami, se kterými jsme se seznámili v předchozích lekcích (nelze zadávat vzorce pro výpočet). Do tabulky můžete zadat hodnoty pro zobrazení grafu. Opakovaným kliknutím na editační ikonu se zadávací tabulka ukryje a dojde k výpočtu a zobrazení grafu dle zadaných hodnot.

Pro změnu typu grafu je v horní liště okna s grafem několik ikon. Každá z nich přepne zobrazení grafu dle svého typu. Další ikona slouží pro přepínání zobrazení dvourozměrného a třírozměrného grafu. Vyzkoušejte si možnosti zobrazení typu grafu a závislost na změně hodnot grafu.

Při zobrazování grafu se propočítává automaticky nejvhodnější zobrazení tak, aby byl graf na vymezené ploše co nejlépe zobrazen. Se změnou hodnot grafu dochází automaticky i ke změně měřítka zobrazení grafu. Kliknete-li pravým tlačítkem myši na plochu grafu, zobrazí se vám zobrazené hodnoty.

Další možností nastavení grafu je zadání popisu os grafu. To je možné provést buď přímo z programu, případně v dialogovém okně, které se zobrazí po kliknutí na editační ikonu s popisem grafu umístěnou v horní liště okna s grafem. Zadat je možné horní, dolní, pravý a levý popis. Po kliknutí na sousední ikonu máte možnost nastavit font pro popis grafu. Legendu grafu (popis základní vodorovné osy) je možné zadat pouze z programu procedurou ChartLegend.

60. lekce - Tisk grafu, uložení do souboru na disk

Vytvořený graf je možné vytisknout, případně uložit do souboru pro další použití.

Před tiskem grafu je vhodné si nastavit parametry tisku v dialogovém boxu, který se zobrazí po kliknutí na ikonu nastavení tisku umístěnou na horní liště okna s grafem. Zadává se umístění vytištěného grafu na stránce. Umístění je možné provést buď myší posouváním předlohu grafu na zobrazení stránky nebo přímým zadáním hodnot. Hodnoty pro umístění grafu od horní, dolní, pravé a levé strany listu se zadávají v palcích (palec = 25.54 mm).

Při definici tisku musíte dbát na to, že pokud zadáte malé rozměry plochy pro tisk, vytiskne se graf postupně na několika stranách. To se dá na druhou stranu někdy výhodně použít, pokud budete potřebovat tisk grafu rozdělit.

Zobrazený graf je možné uložit včetně hodnot do souboru k možnému dalšímu použití. Graf se ukládá na disk do souboru typu *.CHF. Systém KLONDAIK umožňuje uložení grafu na disk a jeho opětné načtení. U grafu, který byl načten ze souboru je možné změnit typ grafu a popisy, není však možné provádět změny hodnot.

61. lekce - Procedury a funkce pro práci s grafy

Systém obsahuje několik procedur a funkcí, sloužících k práci s grafy. Některé možnosti, které jsou přístupné z okna s grafem není přitom možné ovládat z programu. Naopak, některé funkce pro nastavení grafu jsou přístupné pouze z ovládacího okna s grafem. Předpokládá se přitom, že program slouží hlavně k výpočtu a zadání hodnot grafu. Konečné úpravy vzhledu grafu se potom následně nastaví přímo v okně s grafem.

Nejprve by jste měli graf inicializovat procedurou ChartInit, to je zadat počet hodnot na jednotlivých osách. Hodnoty grafu se zadávají procedurou ChartData. Zadané hodnoty se stanou aktivní až po volání ChartUpdate, kdy dojde k přepočítání vzhledu grafu. Typ zobrazovaného grafu se zadává procedurou ChartType, třírozměrnost grafu se nastaví procedurou Chart3D.

Popis grafu (horní, dolní, pravý a levý nadpis) se zadává procedurou ChartTitle. Legendu grafu pro popis hodnot na ose x můžete zadat procedurou ChartLegend. Procedurou ChartGrid máte možnost definovat, zda se má v grafu zobrazovat vodorovná a svislá mřížka sloužící pro snadnější odečet hodnot grafu. Počet zobrazovaných desetinných míst v grafu můžete zadat procedurou ChartDecimals.

Hodnoty grafu můžete uložit do souboru na disk k pozdějšímu použití procedurou ChartSave. Zpětné načtení grafu ze souboru provedete procedurou ChartLoad. Pokud je graf načten ze souboru, není možné již pracovat přímo s jeho hodnotami. Je možné pouze měnit jeho vzhled a nastavení popisu grafu.

```
PROCEDURE main
   VAR
      r : integer
      s : integer
      X : integer
      Y : integer
      Z : integer
   ENDVAR

   r := 4;
   s := 5;
   ChartInit(r, s);
   FOR X := 1 TO R;
       FOR Y := 1 TO S;
           ChartData(x,y,random(300)/3);
       ENDFOR;
   ENDFOR;
   ChartUpdate;
   ChartTitle(0,'LEVÝ POPIS')
   ChartTitle(1,'PRAVÝ POPIS')
   ChartTitle(2,'HORNÍ POPIS')
   ChartTitle(3,'DOLNÍ POPIS')
   ChartLegend(1-1,'leden')
   ChartLegend(2-1,'únor')
   ChartLegend(3-1,'březen')
   ChartLegend(4-1,'duben')
   ChartLegend(5-1,'květen')
   ChartShow;
```

```
ENDPROC;
```

62. lekce - Ovládání oken z programu

Systém KLONDAIK se skládá z několika oken, které slouží k zobrazování výsledných hodnot a ke komunikaci s uživatelem. Každé okno má své specifické určení. Pokud by jste ponechali všechna okna na ploše otevřená, mohli by jste brzy ztratit přehled o jejich obsahu. Proto se nyní seznámíme s tím, jak je možné ovládat okna přímo z programu.

Počet a obsah oken je pevně definován pevně v systému. Není možné žádné okno zrušit. Okna je pouze možné minimalizovat, kdy zabírají pouze malou část plochy. Okna můžete z programu uzavřít jednotlivě procedurou ConsoleHide, ImageHide, ProgramHide, CommandHide, ChartHide, SpreadHide a DbfHide. Pokud potřebujete uzavřít najednou všechna okna, můžete od verze 2.01 systému použít proceduru FormAllHide. Původní velikost oken obnovíte procedurou ConsoleShow, ImageShow, ProgramShow, CommandShow, ChartShow, SpreadShow, DbfShow.

Z programu máte také možnost nastavit polohu a velikost používaných oken. K tomu můžete použít procedury ConsoleForm, ImageFormIMAGEFORM, ProgramForm, CommandForm, ChartForm, SpreadForm a DbfForm. Jako parametry uvedených procedur uvedete vždy levý horní roh okna a šířku a výšku okna v bodech.

Výše uvedenými procedurami máte možnost ovládat z programu nastavení a zobrazení používaných oken. Bude vhodné, pokud si zvyknete na začátku programu nejprve všechna okna uzavřít. Okno, které budete v programu používat potom nastavíte na požadovanou velikost. Jediné okno, které není možné takto ovládat je okno pro výpis obsahu proměnných. Je to proto, že by nebylo vhodné, aby se při ladění programu uzavíralo.

63. lekce - Vytváření nabídek programu

V předcházejících lekcích jsme se naučili používat mnohé vlastnosti systému KLONDAIK včetně jeho knihoven procedur a funkcí. Jistě jste již schopní řešit základní úlohy zadaného problému. Možná by vám proto přišla vhod možnost poskládat již vyřešené programy (úlohy) do jednoho většího programu za použití menu. Ukážeme si proto nyní možnost, jak již sestavené a odladěné programy spojit v jeden kompaktní celek do uživatelsky definovaného menu.

Uživatelské menu nahradí menu systému KLONDAIK včetně jeho ladícího prostředí. Jednotlivé položky uživatelského menu přitom představují volání dosavadních programů. To znamená, že při výběru položky z uživatelsky definovaného menu se spustí program zadaný při definici menu. Obdobně je možné definovat uživatelsky i obsah toolbaru (řada ikon umístěná pod menu). Každá ikona představuje opět zadaný program, který se spustí po kliknutí myší na ikonu.

Uživatelské menu může obsahovat maximálně šest hlavních položek, z nichž první může mít deset podřízených voleb (položek), ostatní potom mohou obsahovat pouze pět podřízených voleb. Dále je možné použít pro volání programů deset uživatelských ikon v toolbaru. Celkem je tedy možné volat z uživatelského menu až 45 uživatelem zadaných programů.

Definici uživatelského menu je vhodné umístit do samostatného programu, který již dále neobsahuje žádný další kód programu. Je to proto, že tento program má za úkol pouze nastavit menu dle vlastní definice a potlačit zobrazování prostředků k ladění programu.

Uživatelské menu se inicializuje (nuluje) voláním procedury MenuInit. Pokud by jste takové menu aktivovali, obsahovalo by pouze jednu volbu sloužící k ukončení uživatelské nabídky s návratem do menu systému KLONDAIK.

Položky uživatelského menu se zadávají procedurou MenuChange. Zadává se přitom úroveň menu, popis položky v menu, volaný program a nápověda k položce menu, která se zobrazuje ve stavovém řádku.

Nadefinované menu se aktivuje voláním procedury MenuActive. Až v tomto okamžiku dojde k náhradě systémového menu za menu uživatelské. Je proto vhodné, aby bylo volání procedury MenuActive jako poslední volaný příkaz v programu.

Následuje příklad programu pro definici uživatelského menu:

```
PROCEDURE main
   MenuInit;
   MenuChange(1,0,'Databáze','',
                  'Příklady obsluhy databází');
   MenuChange(1,1,'Hledání','dbfseek.ips',
                  'Hledání v databázi');
   MenuChange(1,2,'Výmaz','dbfdele.ips',
                  'Výmaz v databázi');
   MenuChange(1,3,'Filtr','dbffilt.ips',
                  'Nastavení filtru');
   MenuChange(2,0,'Formuláře','',
                  'Příklady práce s formuláři');
   MenuChange(1,1,'Ukázka','form2.ips',
                  'použití formulářů');
   MenuChange(1,2,'Hra','form3.ips',
                  'Jednoduchá hra na formuláři');
```

```
    MenuIcon(1,'ruka_v.bmp','domecek.ipx',
                    'Nakreslí domeček');
    MenuActive(true);
ENDPROC
```

Jak již bylo uvedeno, výše uvedený program pouze změní menu systému KLONDAIK dle zadání uživatele a potlačí použití ladících a editačních funkcí systému. Je možné pouze spouštět zadané programy. Pokud se v programu vyskytne chyba, bude vypsáno chybové hlášení a program bude ukončen. Stále však zůstanete v definovaném menu. Uživatelské menu deaktivujete příslušnou volnou z menu, kterou nemusíte definovat a není možné ji také z menu odstranit.

Pomocí dobře navrženého menu aplikace je potlačena zdánlivá nevýhoda možnosti zápisu programů o délce maximálně 32 Kb zdrojového kódu. Uvedená vlastnost nutí tvůrce programu rozčlenit si celkový problém na několik na sobě nezávislých celků, které je možné potom spojit do jednoho celku pomocí uživatelsky definovaného menu.

Tip pro zkušené uživatele:

Nadefinujte si běžným způsobem uživatelské menu, které aktivujete. V podřízených programech můžete následně změnit obsah menu, pokud nepoužijete jeho novou inicializaci, ale provedete pouze změnu obsahu položky menu například podle stavu prováděných výpočtů. To vám umožní měnit dynamicky nejen popis položek menu, ale také programy spuštěné z menu. Zatím však neexistuje způsob, jak položku z menu odstranit.

64. lekce - Kódování programů

Systém KLONDAIK 2 je určen především pro tvorbu programů pro vlastní potřebu. Nepředpokládá se tedy prodej vyvinutých programů dalším zájemcům. Protože je systém interpret a interpretace se provádí přímo nad zdrojovým kódem programu, není jeho zdrojový kód žádným způsobem chráněn proti případným zásahům. Proto byl systém doplněn možností program zakódovat do nečitelné podoby. Původní program zůstane zachován beze změny, zakódovaný program však již není možné dále upravovat ani ladit. Je přitom možné mít na počítači pouze zakódované programy a ty běžným způsobem spouštět bez přítomnosti jejich zdrojového kódu. Takto je možné předat programy dalším zájemcům bez zdrojového kódu.

Programy je možné zakódovat pouze v plné verzi systému (ne v demoverzi)! V zakódovaném programu je uložena informace o sériovém čísle systému KLONDAIK, na kterém program vznikl a při jeho kódování je možné také doplnit i adresu autora programu. Uložené sériové číslo a adresa autora slouží takto jako elektronický podpis tvůrce.

Velkou výhodou zakódovaných programů je, že je lze spustit i v demoverzi systému bez omezení délky programu na 50 řádků (omezení demoverze). Současně je také po dobu provozu zakódovaného programu pozastaven časový limit demoverze. Díky tomu je možné použít demoverzi systému KLONDAIK jako runtime pro provozování obsáhlejších zakódovaných programů.

Pro zakódování programu je určena volba menu Program/Zakódovat. Zobrazí se dialogové okno, ve kterém máte možnost zadat adresu, která bude uložena v zakódovaném programu (také v nečitelné podobě). Zakódováním programu vznikne nový soubor stejného jména jako program, ale s příponou *.IPX.

Zakódovaný program je možné načíst běžným způsobem, kdy se v dialogovém okně pro výběr souboru zadá maska *.IPS. Po načtení zakódovaného programu není zobrazován jeho zdrojový kód, není také možné program vytisknout, ale také krokovat apod. Spouštění zakódovaných programů je stejné jako u běžných programů.

6.0. JAZYK INTER-PASCAL

Protože je zabudovaný programovací jazyk v programu OZOGAN KLONDAIK v mnnohém podobný standardnímu jazyku Pascal, nazvali jsme jej INTER-PASCAL. Názvem jazyka jsme chtěli zdůraznit, že se jedná o interpretovaný jazyk, podobný jazyku Pascal.

Standardní jazyk Pascal je kompilátor, což znamená, že zdrojový text programu je přeložen do spustitelné EXE podoby, kterou zpracovává přímo mikroprocesor počítače. Díky tomu je kompilovaný program rychlý a nepotřebuje již ke své činnosti žádnou další podporu. Chceme-li udělat v programu úpravy, musí se program upravit a znovu zkompilovat. Ladění kompilovaných programů je pro začátečníka složitější a nelze jednoduchým způsobem vyzkoušet funkci jednotlivých příkazů a procedur.

Jazyk INTER-PASCAL je zpracováván interpretem, což znamená, že po spuštění programu je každý řádek zdrojového textu interpretován na příslušné příkazy mikroprocesoru počítače, který potom vykonává veškeré výkonné akce. Nevýhodou je, že vykonávání interpretovaného programu je díky tomu pomalejší. Velikou výhodou je ale velmi jednoduché ladění programu a možnost zkoušení příkazů a procedur programu v příkazovém okně přímo v prostředí interpretu bez nutnosti kompilace programu.

Jazyk Pascal byl navržen počátkem sedmdesátých let profesorem N.Wirthem. Při definici jazyka vycházel ze zásady strukturového programování a sledoval hlavní cíl, kterým bylo vytvořit jazyk vhodný pro výuku programování. V tehdejší době však byly počítače mohutná a složitá zařízení umístěná v klimatizovaných sálech. Programy se do počítače zadávaly pomocí děrných štítků nebo děrné pásky. Celkově se proto předpokládalo, že i programy v jazyce Pascal budou na rozdíl od dnešního masového rozšíření počítačů i do mnoha domácností sestavovat především odborníci.

Postupným vývojem výpočetní techniky byly do jazyka Pascal přidávány další možnosti. Dnešní překladače jazyka Pascal jsou proto dodávány včetně dokumentace obsahující tisíce stran a pro jejich plné zvládnutí je nutné dlouhodobé používání. Proto byl navržen jazyk INTER-PASCAL, který šel opačnou cestou, tedy zjednodušení základní definice jazyka Pascal tak, aby jej mohli používat i běžní, neprofesionální uživatelé. Je proto určen především pro výuku základů programování ve školách, uživatelům domácích počítačů a podobně. Jeho hlavním cílem je umožnit programovat v prostředí Windows i neprofesionálním uživatelům.

7.0. VLASTNOSTI JAZYKA INTER-PASCAL

Jazyk INTER-PASCAL je podmnožinou standardního jazyka Pascal. Obsahuje však některé změny a úpravy, díky kterým je v mnoha případech jednoduší a snažší. Pokud znáte základy jazyka Pascal, nebude pro vás problém naučit se používat jazyk INTER-PASCAL. Naopak, po zvládnutí jazyka INTER-PASCAL bude pro Vás snadné přejít na standardní zápis programů v jazyce Pascal.

Jazyk INTER-PASCAL vychází ve své syntaxi, obsažených příkazech, funkcích a procedurách z klasického jazyka Pascal. Struktura zápisu programu však byla zjednodušena a v mnohém také zpřehledněna způsobem zápisu struktury programu ve stylu databázových jazyků typu Dbase a FoxPro. Některé drobné prvky jazyka byly převzaty z jazyků Basic a C. Výsledkem by měl být nový jazyk (verze jazyka ?), který může být sice mnohými odborníky a recenzenty odsuzován, běžný uživatel, pro kterého je program určen jej však jistě uvítá.

7.1. Co je to program a jak vzniká ?

Program je definovaná posloupnost příkazů popisujících nějakou činnost. Takovým programem je například i kuchařský recept, podle něhož postupuje kuchař při vaření. Na začátku receptu bývá většinou seznam potřebných surovin. Dále následuje přesný popis jejich zpracování. V popise se přitom často uvádí postupy závislé na splnění určitých podmínek. Například pokud nemáme kečup, lze použít rajčata. Dále může být v receptu definováno opakování určité akce. Například míchat, dokud kaše nezhoustne. Na podobných principech (hodnoty, posloupnost akcí, podmínky a opakování) je založen i počítačový program.

Program v počítači zpracovává zadané hodnoty, to je data. Vlastnostmi dat jsou jejich označení, struktura, hodnoty a možné operace. To vše je definováno deklarací dat. Z hlediska výše uvedeného kuchařského receptu se jedná o seznam potřebných surovin.

Kromě deklarací a příkazů pro zpracování dat se v programu vyskytují také příkazy, jimiž se řídí návaznost akcí při provádění programu v závislosti na hodnotách dat. Pomocí těchto příkazů se specifikuje větvení programu. Další příkazy umožňují opakované provádění určitých příkazů. Opět jako u kuchařských receptů, kde se však již předpokládají základní znalosti vaření. Počítačové programy však zpracovává neinteligentní stroj, kterému musíte zadat přesný postup zpracování. Program si nemůže sám domýšlet co má provést, jestliže nejsou některé činnosti v programu definovány. Psaní programu pro počítač proto vyžaduje od autora systematický přístup a pečlivost. Před zápisem programu by měly předcházet určité přípravné práce.

Nejprve je nutné přesně definovat, co má program řešit. Nepřesná a neúplná definice problému může vést ke špatnému pochopení a tím i k vytvoření chybného programu. Definice problému by proto měla obsahovat specifikaci vstupních dat (jejich strukturu, formát, a možné hodnoty). Dále je nutné obdobným způsobem specifikovat výstupní data. Stejně důležitá je ale také specifikace vyjímečných situací, pokud by například data neodpovídala požadovaným hodnotám.

Dalším krokem vývoje programu je nástin řešení programu a návrh algoritmu zpracování. Složité problémy se snažíme rozložit na jednoduší celky a postupně zjemňujeme specifikaci řešení až na úroveň operací použitelných v programovém jazyce. Nedílnou součástí je přitom postupný návrh datových struktur, s nimiž bude program pracovat.

Po sestavení algoritmu programu následuje tzv. kódování programu, kdy převedeme sestavený algoritmus do zápisu v programovém jazyce. Složitost kódování závisí přitom na kvalitě a členění sestaveného algoritmu řešení.

Poslední, velmi důležitou částí tvorby programu je ověření jeho funkčnosti. Program, který poskytuje výsledky nemusí být totiž vždy správný. Musíme proto program ověřit na vhodné množině zkušebních vstupních dat, pro něž známe správné výsledky. Mnohdy se například zapomíná ošetřit stav, kdy se zadají nepředpokládané vstupní údaje a program skončí chybou z důvodu dělení nulou.

7.2 Základy zápisu programů

Program v jazyce INTER-PASCAL je textový soubor libovolného názvu s příponou *.IPS. Délka souboru s programem je omezena na 32 kB. Pro zápis programu není předepsán pevný formát. Přesto je však vhodné dodržet několik zásad, díky kterým bude zápis programu přehlednější. Vzorem zápisu mohou být například programy dodávané spolu s programem INTER-PASCAL.

Jednotlivé řádky programu smí mít délku maximálně 255 znaků. Vzhledem k přehlednosti programu je však vhodné dodržovat délku řádku zhruba do 80 znaků. Nejvhodnější je zapisovat každý příkaz na samostatný řádek. V jazyce Pascal se musí jednotlivé příkazy oddělovat znakem středník. Jazyk INTER-PASCAL to sice nevyžaduje, přesto však doporučujeme toto pravidlo zachovat. Pouze za situace, kdy budete chtít napsat na jeden řádek více příkazů, musíte toto pravidlo bezpodmínečně dodržet. Proto by bylo možné v případě nutnosti naprogramovat na jeden řádek programu následující sérií příkazů:

```
FOR x := 1 to 100; y[x] := 0; ENDFOR
```

Na řádcích programu nelze rozdělovat mezerou nebo novým řádkem jména příkazů, proměnných procedur a funkcí. Naopak, je nutné oddělit minimálně jednou mezerou nebo novým řádkem jednotlivé příkazy, jména proměnných, procedur a funkcí. Řetězce vymezené uvozovkami nebo apostrofy není možné rozdělit novým řádkem. V případě nutnosti je možné použít spojování řetězců znaménkem plus.

V programu nejsou rozlišována velká a malá písmena. Lze proto psát velkými i malými písmeny. Jedinou vyjímkou jsou řetězce uzavřené mezi dvěma uvozovkami nebo apostrofy. Přesto se však doporučuje ke zvýšení přehlednosti zápisu dodržovat jednotný styl zápisu. Názvy příkazů psát velkými písmeny, jména proměnných, funkcí a procedur začínat velkým písmenem. V případě složení jména z několika slov neuvádět mezi slovy mezery a každé slovo začít velkým písmenem.

Pro zvýšení srozumitelnosti zdrojového programu je možné uvádět v něm komentáře. Komentář je posloupnost libovolných znaků uzavřená mezi složené závorky a neobsahující další složené závorky. Takto lze například dočasně označit část zdrojového textu programu, která se nemá vykonávat. Je možné označit jedním párem složených závorek najednou i několik řádek programu. Další, nestandardní možnost (v jazyce Pascal nelze) je uvedení dvojice lomítek, které způsobí označení dalšího textu až do konce řádky za komentář. Nelze použít vymezení komentáře dvojicí znaků (* pro počátek a *) pro konec komentáře, jak je obvyklé u jazyka Pascal.

Příklad:

```
// komentář podle jazyka C++
FOR x := 1 to 100  {step 5};  {dočasné zrušení kroku}
    y[x] := 0;     {libovolná poznámka}
{   Point(x, y, 2)
    MoveTo(x+5, y+5)
}
ENDFOR             // takto lze také označit komentář
```

Pro zvýšení přehledu je vhodné dodržovat určitou úpravu zápisu programu s vyznačením strukturovaného zápisu programu u složených příkazů. To znamená, že příkazy uvnitř cyklů a podmínek budou posunuty o několik mezer. Díky tomu lze jednoduše a přehledně sledovat strukturu vykonávání příkazů v cyklech a rozhodovacích podmínkách.

Příklad:

```
FOR x := 1 to 100;
    y[x] := 0;
    IF x < 50 then
       Point(x, y, 2);
       MoveTo(x+5, y-5);
    ELSE
       Point(x, y, 2);
       MoveTo(x-5, y+5);
    ENDIF
ENDFOR
```

Programy jsou v následujících textech vytištěny jednotným typem neproporcionálního písma. Pro možnost znázornění variant zápisu je použito několik symbolů, které se při skutečném zápise programu nahradí skutečným zápisem. Pokud jsou v příkladech uvedeny tři tečky za sebou, znamená to, že lze místo nich uvést libovolné příkazy. Je-li v zápisu syntaxe uvedena část zápisu v hranatých závorkách, je uvedená část nepovinná, nemusí být uvedena. Pokud je možný výběr z několika variant, jsou jednotlivé volby odděleny svislou čarou. Uvedené konstrukce musí být v konkrétním zápisu programu vždy nahrazeny skutečným zápisem, jinak bude při provádění programu nahlášena chyba v programu !

Příklad:

```
IF x > 10 [then]    {klíčové slovo then nemusí být uváděno}
   ...              {možnost zápisu libovolných příkazů}
ELSE
   Inc(x [, 5])     {zvýšení hodnoty pouze volitelně}
ENDIF
```

7.3. Struktura programu

Standardní jazyk Pascal dodržuje přísný formát zápisu programu. Obecně platí pravidlo, že každý prvek programu se musí nejprve deklarovat a až potom se může používat. Na rozdíl od toho je možné v jazyce INTER-PASCAL uvádět deklarace kdekoliv. Přesto však doporučujeme zachovat dále uvedený postup, kdy nejprve nadefinujete proměnné a záznamy. Dále je vhodné uvést uživatelsky definované funkce a procedury. Až na konci programu zapište hlavní výkonnou proceduru programu, která musí mít jméno main.

```
TYPE
   {definice datových typů - záznamů}
ENDTYPE

VAR|GLOBAL|LOCAL
   {deklarace proměnných}
ENDVAR

PROCEDURE ...
   {deklarace uživatelské procedury}
ENDPROC

FUNCTION ...
   {deklarace uživatelské funkce}
ENDPROC

PROCEDURE main
   {tělo hlavního programu}
ENDPROC
```

V programu se může vyskytovat několik bloků s definicí procedur a funkcí. Jména procedur a funkcí však musí být jednoznačná a nesmí se v rámci programu opakovat. V programu musí být vždy definována procedura se jménem main, kterou doporučujeme umístit vždy na konec programu. Procedura main je hlavní procedurou celého programu, protože jejím prováděním se začíná vždy činnost programu.

V rámci definice procedury a funkce může být vnořena deklarace proměnných. Takovéto proměnné mají potom pouze lokální (místní) platnost a jsou přístupné pouze v proceduře a funkci, ve které byly deklarovány.

7.4. Definice typů (záznamů)

Pro výpočty v programu se používají hodnoty, které musí být vždy definovaného typu. Typy dat charakterizují obory hodnot proměnných a příslušné operace nad těmito hodnotami. Mezi běžné typy dat patří například typ celých čísel (integer), typ reálných čísel (real) nebo typ logických hodnot. Tyto typy jsou již dále nedělitelné. Jazyk INTER-PASCAL, stejně jako klasický jazyk Pascal umožňuje používání nejen předdefinovaných typů, máte také možnost nadefinovat své vlastní nové typy. Nově definované typy se přitom mohou skládat nejen z předdefinovaných typů (základní definice proměnných), ale i z dříve definovaných vlastních typů.

Vlastní definované typy se většinou sdružují do strukturovaných údajů které se nazývají záznamy. To je výhodné například pro definici data, kde sdružíme položky datum, měsíc a rok do jednoho záznamu. Obdobně je možné definovat nový typ pro adresu nebo například osobní údaje. To nám následně umožní přistupovat ke každé složce záznamu samostatně, umožní nám ale také přistupovat k záznamu jako k jedné proměnné. Pro definici typu záznam musíte použít příkazy s následující syntaxí:

```
TYPE
   myNewType = RECORD
      Field1 : Field1Type;
      Field2 : Field2Type;
   ENDRECORD
ENDTYPE
```

Pokud potřebujete zapsat deklaraci většího počtu struktur typu záznam, můžete tak učinit v rámci jednoho bloku TYPE .. ENDTYPE. Záznamy nesmí být deklarovány v procedurách. Nejvhodnější je definovat je ihned na začátku programu, ještě před definicí procedur a funkcí. Záznamy mají proto vždy globální platnost.

Pro definování proměnné typu záznam použijte v programu standardní deklarační metodu s uvedením typu záznam.

```
VAR
   MyRecordVar : MyNewType
ENDVAR
```

Pro přístup k jednotlivým polím typu záznam musíte použít pro definici jména název záznamu a název pole oddělené tečkou:

```
MyRecordVar.field1 := 'abc'
```

Kompletní příklad použití:

```
TYPE
   Adresa   = RECORD
      firma : string;
      ulice : string;
      mesto : string;
   ENDRECORD
ENDTYPE

VAR
   Zakaznik : Adresa;   {definice proměnné typu záznam}
```

```
ENDVAR
...
Zakaznik.firma := 'OZOGAN';      {zápis hodnot}
Zakaznik.ulice := '1. máje 97';
Zakaznik.mesto := 'Liberec';
...
writeln('firma: ', Zakaznik.firma); {čtení hodnot záznamu}
```

Jazyk INTER-PASCAL umožňuje pracovat pouze se soubory typu text. Textové soubory je možné zpracovávat pouze sekvenčně, to znamená postupně od začátku do konce. Při tomto způsobu není možné vyhledávání podle klíčů. Proto byl jazyk doplněn o několik procedur a funkcí s možností zpracování záznamů pomocí vyhledávacích indexů. To Vám umožní zpracovávat záznamy s možností vyhledávání požadovaných záznamů pomocí klíčů. K tomuto účelu je nutné definici záznamu doplnit o označení klíčových položek klíčovým slovem KEY:

```
TYPE
   myRecord = RECORD
      KEY key1 : key1Type;
      KEY key2 : key2Type;
      ...
      field1   : field1Type;
      field2   : field2Type;
   ENDRECORD
ENDTYPE
```

V záznamu myRecord jsou deklarovány čtyři položky. Dvě z nich, označené klíčovým slovem KEY mohou sloužit pro vyhledávání záznamů.

Definice proměnné typu tabulka:

```
VAR
   MyTable : table of myRecord;

ENDVAR
```

Jazyk INTER-PASCAL obsahuje v knihovně pro zpracování záznamů několik procedur a funkcí, které jsou použitelné pro vyhledávání, záznam, aktualizaci, čtení a mazání záznamů s použitím klíčových položek.

7.5. Deklarace proměnných

Proměnné jsou prvky programu, které v programu ukládají do paměti hodnoty výpočtů k dalšímu použití. Proměnné mohou být číselné, znakové, řetězcové nebo logické. Typ proměnné je dán typem definujícího výrazu.

Deklaraci proměnných uvádí klíčové slovo VAR. Pro rozlišení lokálních a globálních proměnných lze používat i klíčová slova LOCAL a GLOBAL. Konec definice bloku proměnných definuje klíčové slovo ENDVAR.

Deklarace obsahuje seznam identifikátorů, které označují nové proměnné a jejich typ. Typ proměnné může určovat identifikátor typu, definovaný dříve v úseku definic typů ve stejném bloku, nadřazeném bloku. K vyjádření typu lze použít přímo klíčového slova, které vyjadřuje jeden ze standardních typů (Integer, Byte, Real, String,...).

Na rozdíl od standardního jazyka Pascal nelze definovat najednou několik proměnných ve tvaru: x, y : integer. Je ale možné definovat více proměnných na jednom řádku s uvedením typu pro každou proměnnou s oddělením proměnných středníkem. Problémy mohou také nastat, pokud se budete snažit zneplatnit definici proměnné příznakem pro komentář ve tvaru dvou lomítek, což v definici proměnných nelze. Je možné ale použít standardní komentář ve složených závorkách. Například:

```
VAR
   a    : integer
   b    : integer; c : string
   c, d : integer  { takhle nelze, bude hlášena chyba !!!}
// e    : integer  { v definici proměnných nelze !!!}
{  f    : integer    toto ale možné je !!! }
ENDVAR
```

Deklarace proměnné má platnost v bloku, kde byla deklarována. Pokud je proměnná deklarována na začátku programu, před deklaracemi procedur a funkcí, lze se na proměnnou odvolávat ve všech procedurách programu. Pokud je deklarace proměnné uvedena ve vnořené proceduře nebo funkci, pak se lze na proměnnou odkazovat pouze v rámci této procedury nebo funkce. Deklarace proměnné ve vnořeném bloku, která se jmenuje stejně jako proměnná v nadřazeném bloku, nezpůsobí změnu hodnoty proměnné v nadřazeném bloku.

Proměnné deklarované vně procedur a funkcí se nazývají globální. Proměnné deklarované uvnitř procedur a funkcí se nazývají lokální.

Odkaz na proměnnou se provádí identifikátorem proměnné. Proměnná má platnost v bloku programu, kde byla deklarována. Pokud se provádí odkaz na strukturovanou proměnnou (typu záznam), může se definovat prvek struktury.

Když deklarujeme proměnnou, musíme určit její typ. Typ proměnné popisuje množinu hodnot, které může nabývat a operace, které se na ní mohou provádět. Definici typu specifikuje identifikátor, který označuje typ. Když se identifikátor vyskytuje na levé straně definice typu, je definován jako identifikátor typu pro blok, ve kterém je definice uvedena. Identifikátor na pravé straně definice je odkazem na předem definovaný typ.

Jazyk INTER-PASCAL připouští zadávání číselných hodnot v hexadecimálním formátu. Ke specifikaci hexadecimální hodnoty je nutné před vlastním vyjádřením hodnoty vložit znak '$'.

Příklad:

```
$24       hexadecimální vyjádření
 49       dekadické vyjádření
```

Proměnné v jazyce INTER-PASCAL musí být deklarovány ještě dříve, než budou poprvé použity. Proměnné jsou buď globální, ke kterým může přistupovat každá procedura (mají globální rozsah platnosti), nebo lokální s platností pouze v proceduře/funkci, ve které byly definovány.

7.6. Typy proměnných

Základní proměnné jsou nejjednodušší proměnné, které reprezentují jeden výskyt prvku údaje. Ze základních typů proměnných jsou odvozovány další deklarace proměnných typů pole, záznam a tabulky.

Podporované typy proměnných:

BYTE	- celočíselný typ s rozsahem 0 až 255
INTEGER	- celočíselný typ s rozsahem od -32768 do +32767
WORD	- celočíselný typ s rozsahem od 0 do 65535
LONGINT	- celé číslo od -2 miliard do + 2 miliard
REAL	- číslo v rozsahu od 2.9*10-39 do 1.7*1038
STRING	- řetězec znaků o délce 255 znaků
CHAR	- jeden znak
BOOLEAN	- logická proměnná typu BOOLEAN (True/False)
TEXT	- textový soubor typu Pascal TEXT
ARRAY	- jednorozměrné pole proměnných

Ačkoliv jazyk INTER-PASCAL nepodporuje všechny typy jazyka Pascal, jsou klíčová slova všech ostatních typů rezervovaná pro možné budoucí rozšíření jazyka.

Typ BYTE zabírá v paměti jednu slabiku a může proto obsahovat pouze celá čísla pro hodnoty od 0 do 255. Lze jej proto využít pro počítadlo řádků na stránce apod.

Datový typ INTEGER je v paměti uložen ve dvou slabikách a může proto nabývat hodnot od -32768 do +32767.

Typ WORD zabírá v paměti také dvě slabiky, ale protože může obsahovat pouze kladná čísla, může obsahovat čísla od 0 do 65535.

Typ LONGINT může nabývat záporných hodnot a protože obsahuje v paměti čtyři slabiky, může nabývat hodnot od mínus dvou miliard do plus dvou miliard.

Datový typ REAL se používá pro vyjádření hodnot s pohyblivou řádovou čárkou. Číslo je uloženo v šesti slabikách a může nabývat hodnot od 2.9*10-39 do 1.7*1038

Datový typ znak CHAR má velikost jedné slabiky a uchovává jeden znak. Znakové konstanty se vyjadřují tak, že se znak uzavře do apostrofů, například 'A', '1', '&'. Zápis '1' v tomto případě znamená znak pro vyjádření číslice jedna nikoli číselnou hodnotu. Na rozdíl od klasického jazyka PASCAL je možné znaky uzavírat i do uvozovek.

Datový typ STRING představuje posloupnost znaků s pevnou délkou 255 znaků typu CHAR bezprostředně za sebou. Vytváří tím tzv. řetězec. Řetězec musí být ohraničen apostrofy nebo dvojitou uvozovkou. Řetězec může obsahovat maximálně 255 znaků. a je možné k němu přistupovat jako k jednorozměrnému poli. Příklad:

```
s := 'OZOGAN'            {naplní řetězcovou proměnnou}
IF s[2] = 'Z'            {pokud je druhý znak písmeno Z}
   WRITELN(length(s))    {vypíše délku řetězce}
ENDIF
```

Pozor, ve standardním jazyce Pascal je na nulté pozici uložena celková délka řetězce. Toto

pravidlo jazyk INTER-PASCAL nepodporuje, pro zjištění délky řetězce můžete použít funkci length.

Velice často se v programech používá tzv. prázdný řetězec, který mezi apostrofy nic neobsahuje. Pokud chceme do řetězce vložit znak apostrof, musíme uvést dva bezprostředně následující apostrofy. Ve standardním jazyce Pascal mohou být do znakových řetězců vloženy i řídící znaky, což jazyk INTER-PASCAL nepodporuje.

Datový typ BOOLEAN má pouze dvě možné hodnoty a to True (pro stav pravda) a False (pro stav nepravda). Lze mu proto v programu přiřadit buď konkrétní hodnotu True nebo False, případně výsledek vyhodnocení logického výrazu. Logické výrazy najdou uplatnění zejména v podmíněných příkazech, příkazech cyklu apod.

Proměnná typu TEXT umožňuje zaznamenávat data do souboru. Soubor je textového typu a umožňuje záznam posloupnosti znaků (textových řádků) různé délky. Každý řádek tohoto textu může proto obsahovat různý počet znaků a je ukončen řídícím znakem 'konec řádku'. Textové soubory je možné zpracovávat pouze sekvenčně řádek po řádku bez možnosti přímého přístupu pomocí klíčů.

Proměnná typy ARRAY představuje datovou strukturu položek stejného typu, které se vzájemně odlišují pomocí indexu. Jako index pole slouží hodnoty určitého typu, který nazýváme typem indexu pole a který je spolu s typem složek pole stanoven popisem typu pole. Počet složek pole je dán při deklaraci pole a není vhodné z důvodu velikosti pole v paměti uvádět hodnotu větší než 1024. Typ složky nesmí být vlastní definovaný typ, případně další pole nebo typ TEXT.

7.7. Proměnné typu pole

Programy v jazyce INTER-PASCAL mohou definovat a používat proměnné typu pole. V aktuální verzi programu je možné definovat pouze pole jednorozměrné (jedna dimenze). Pole jsou deklarována použitím následující syntaxe:

```
VAR
   MyArrayVar: array[low..high] of <typ>;
ENDVAR
```

Kde <typ> je typ proměnné, ze kterého se pole skládá. Low a High definují dolní a horní index pole.

Příklad:

```
VAR
   vyrobek: array[1..16] of string;
ENDVAR
```

Pole vyrobek je v tomto případě definováno jako jednorozměrné pole s šestnácti prvky s indexy 1 až 16. Typ prvků pole je řetězec. Přistupovat k prvkům pole je možné pomocí indexů pole s následující syntaxí:

```
MyArrayVar[Index] :=     ... { přiřazení do prvku pole }
xxx := MyArrayVar[Index] ... { přistup k prvkům pole}
```

Index je celočíselný výraz který definuje přesnou pozici prvku v poli. Všimněte si, že index musí mít menší hodnotu low a větší hodnotu high podle deklarace pole.

Výše uvedený druh pole má ihned při jeho deklaraci znám počet indexů pole. Dalším druhem pole jsou tzv. otevřená pole, která jsou podporována jako parametry funkcí a procedur. Tato pole nemají ve své deklaraci uveden rozsah (počet položek). Pro definici otevřeného pole se používá následující syntaxe:

```
TheOpenArray: array of <type>
```

Kde <type> je typ prvků v poli.

Pro převod parametrů do procedury, která očekává otevřené pole se používá následující syntaxe:

```
[element1, element2, .., elementN]
```

Program může přistupovat k prvkům pole za použití obvyklé synataxe:

```
proměnná[index]
```

Standardní knihovna jazyka INTER-PASCAL definuje funkci Low a High, které se mohou použít pro zjištění rozsahu dolní a horní hranice otevřeného pole.

Viz také:
funkce Low, funkce High

7.8. Deklarace VAR .. ENDVAR

```
VAR|LOCAL Var-Name:Var-Type [= Initialization-Value] [;]
   [ Var-Name... ]
ENDVAR
```

Deklarace se používá pro definici lokální (místní) proměnné použitelné pouze v rámci procedury, ve které byla proměnná definována. Tato proměnná není přístupná z jakékoliv jiné procedury. Proměnnou je možné v proceduře definovat pouze jednou, protože interpret by nebyl schopen rozpoznat, která definice je platná!

Var-Name v definici je jméno nově definované proměnné, Var-Type je typ proměnné, a volitelná Initialization-Value může obsahovat hodnotu definované proměnné.

Vzhledem k tomu, že INTER-PASCAL je interpretovaný jazyk, může volitelná definice hodnoty proměnné obsahovat například i matematický výraz.

Příklad:

```
VAR a1 : integer
    a2 : string
    a3 : integer = 12*22
ENDVAR
```

Pro jednodušší rozlišení platnosti deklarovaných proměnných lze místo klíčového slova VAR použít plně kompatibilní (náhradní) klíčové slovo LOCAL. Pro zachování kompatibility se standardním Pascalem je však vhodnější používat VAR. Deklarace však musí být i v tomto případě na rozdíl od jazyka Pascal ukončena příkazem ENDVAR.

Všimněte si, že pokud je klíčové slovo VAR použito v definici procedury, nebo funkce, jsou definované proměnné lokální a platné pouze uvnitř procedury/funkce. Pokud bude ale proměnná definována vně procedury nebo funkce, bude platnost proměnné globální. To je umožněno vlastností interpretu, který si v tabulce symbolů uchová informaci o definici proměnné po celou dobu zavedení zdrojového souboru.

Viz Také:
příkaz GLOBAL, proměnné, výrazy

7.9. Deklarace GLOBAL .. ENDVAR

```
GLOBAL Var-Name : Var-Type [= Initialization-Value] [;]
    [ Var-Name... ]
ENDVAR
```

Deklarace GLOBAL definuje globální proměnnou, která může být zpřístupněná ze všech procedur INTER-PASCALu. Globální proměnné je vhodné definovat vně jakýchkoliv procedur. Proměnnou je možné v proceduře definovat pouze jednou, protože interpret by nebyl schopen rozpoznat, která definice je platná.

Var-Name v deklaraci je jméno nové proměnné, Var-Type je typ proměnné, a ve volitelné Initialization-Value může být specifikovaná hodnota proměnné. Deklarace globálních proměnných musí být ukončena klíčovým slovem ENDVAR.

Příklad:

```
GLOBAL
   b1 : integer
   b2 : string  = 'OZOGAN'
   b3 : integer = 12*22
ENDVAR
```

Viz také:
příkaz LOCAL, proměnné, výrazy

7.10. Procedury a funkce

Procedury a funkce, souhrnně nazvané podprogramy, dovolují vícenásobné používaní napsaného kódu programu. Rozčlení program do přehledných částí a umožní vlastně nové, uživatelsky definované příkazy jazyka.

Rozdíl mezi procedurou a funkcí je v tom, že funkce vrací hodnotu a může se použít přímo ve výrazech. Procedura se volá příkazem volání procedury ke splnění jedné nebo více operací.

Definice procedury:

```
PROCEDURE proc_name[(seznam_parametrů)]
   .. kód procedury
ENDPROC
```

Definice funkce:

```
FUNCTION func_name[(seznam_parametrů)] : návratový_typ
   .. kód funkce
ENDPROC
```

Kde poc-name/func-name je jméno procedury nebo funkce. Toto jméno musí být v celém programu jednoznačné, protože by interpret při vykonávání programu nevěděl, kterou proceduru (funkci) má vykonat.

Volitelný seznam_parametreů je seznam parametrů, které se předají proceduře nebo funkci s využitím následující syntaxe:

```
jméno_parametru:typ_parametru[,jméno_parametru:typ_parametru]
```

Kde jméno_parametru je jméno parametru, který bude použit v proceduře a typ_parametru je typ parametru (integer, string,..).

Návratový_typ v definici funkce je typ výsledku vráceného funkcí.

Na rozdíl od běžného Pascalu lze funkci i proceduru definovat kdekoliv v programu. Není proto nutná deklarace funkce před jejím prvním použitím. Je to proto, že interpret si aktualizuje vnitřní tabulku funkcí a procedur v době zavádění zdrojového textu programu do paměti. Blok příkazů nesmí být ohraničen klíčovými slovy BEGIN a END jako u standardního jazyka Pascal.

Volání procedur a funkcí:

Volání procedur a funkcí je funkčně obdobné volání příkazů jazyka INTER-PASCAL. Pokud je vyvolána procedura nebo funkce, předá se řízení programu specifikované proceduře/funkci a pokračuje se v činnosti programu prováděním instrukcí této procedry/funkce. Po ukončení volané procedury/funkce je proveden návrat do programu za místo volání procedury/funkce.

Volání procedury se provádí následujícím zápisem:

```
procedure_name[(parametr-1, parametr-2 [, parametr-n])]
```

Kde procedure_name je jméno volané procedury nebo funkce, které je definováno v jazyce INTER-PASCAL nebo uživatelem v programu.

Volitelné parametry jsou parametry definované v definici procedury.

7.11. Výrazy

Za výrazy se považují veškeré aritmetické, logické nebo jiné operace. Výrazy se skládají z operátorů a operandů. Operátor slouží k manipulaci s datovými typy, které reprezentují operandy. Operátory mohou být binární, nebo unární.

Většina operátorů v jazyce INTER-PASCAL jsou binární - obsahují dva operandy. Ostatní jsou unární - pouze s jedním operandem. Binární operátory mají běžný algebraický tvar, jsou umístěny mezi operandy. Unární operátor vždy předchází před svým operandem. Operátory mají různou prioritu, která je vyjádřena pořadím v následujícím přehledu (řazeno od nejvyšší priority):

1 - proměnné a funkce vracející numerickou hodnotu
2 - závorky
3 - unární operátory NOT, unární mínus (viz např. -1)
4 - násobící operátory *, /, MOD, AND, SHL, SHR
5 - sčítací operátory +, -, OR, XOR
6 - relační operátory =, <>, >, <, <=, >=

Operand mezi dvěma operátory s různou prioritou je vázán na operátor s vyšší prioritou. Operand mezi dvěma operátory se stejnou prioritou je vázán na operátor po své levé straně. Operace s rovnocennou prioritou se provádějí zleva doprava. Prioritu operací lze upravit pomocí kulatých závorek, vždy se nejprve vyhodnotí výraz v závorkách, bez ohledu na prioritu.

Vzájemné vztahy mezi prvky programu lze vyjádřit pomocí relačních operátorů:

>	větší než
>=	větší nebo rovno než
<	menší než
<=	menší nebo rovno než
=	rovno
<>	nerovno

Relační vztahy mezi prvky programu jsou logické výrazy a výsledkem jejich vyhodnocení je logická hodnota True (pravda) nebo False (nepravda). Není možné tvořit výrazy vyhodnocováním numerických výrazů s výrazy řetězcovými nebo logickými.

Numerické výrazy:

V jazyce INTER-PASCAL jsou všechny numerické výrazy vyhodnocovány jako typ REAL a data jsou konvertována zpět dle deklarace procedury/funkce.

Numerické výrazy podporují standardní matematické operace (+,-,*,/). Možné je také použití závorek a unární mínus.

Základem numerických výrazů jsou numerické konstanty, proměnné numerického typu, a funkce které vracejí numerickou hodnotu. Numerická konstanta je dekadické číslo, pokud má předponu $, jedná se o šestnáctkovou konstantu (viz $10 = 16, $FF = 256).

Řetězcové výrazy:

Primitivní základy řetězcových výrazů jsou řetězcové konstanty, proměnné řetězcového typu, a funkce které vracejí řetězcovou hodnotu.

V jazyce PASCAL je možné vymezit řetězce pouze jednoduchými uvozovkami. V jazyce INTER-PASCAL jsou však řetězcové konstanty vymezeny buď jednoduchými nebo i dvojitými

uvozovkami. Pokud potřebujete v řetězci zobrazit uvozovky, můžete tak učinit uvedením opačného typu. Tak lze například definovat řetězec:

```
"text  'mezi' uvozovkami"
```
nebo
```
'jiný "příklad" uvozovek'
```

Řetězcové výrazy podporují sčítání řetězců za použití operátoru plus. Je možné například použít výrazu "abc"+"def". Při porovnávání řetězců je kritériem uspořádání znaků podle tabulky ASCII. Lze porovnávat dvě libovolné řetězcové hodnoty, protože všechny řetězcové hodnoty jsou kompatibilní. S hodnotou typu řetězec je rovněž kompatibilní hodnota typu Char.

Logické výrazy

Logické výrazy INTER-PASCALu vracejí Boolovskou hodnotu - TRUE (pravda) nebo FALSE (nepravda). Podporované logické operátory jsou AND, OR, XOR, NOT a závorky.

8.0. PŘÍKAZY JAZYKA INTER-PASCAL

:=	- příkaz přiřazení
IF .. ELSE .. ENDIF	- rozhodovací blok podmínky
FOR .. ENDFOR	- cyklus s definovaným počtem opakování
REPEAT .. UNTIL	- cyklus s vyhodnocením podmínky na konci
WHILE .. ENDWHILE	- cyklus s vyhodnocením podmínky na začátku
SWITCH .. ENDSWITH	- vícenásobné větvení činnosti programu
CONTINUE	- přerušení cyklu
READ	- čtení hodnoty z klávesnice, souboru
WRITELN	- výpis obsahu proměnné
RETURN	- ukončení procedury, funkce

Protože se v jazyce INTER-PASCAL nepoužívá standardní oddělení bloků programu konstrukcí BEGIN .. END, jsou složené příkazy ukončeny vždy svým ukončovacím příznakem. Například ENDIF nebo ENDFOR. Program se tím stává mnohem přehlednější. Takto strukturovaný zápis programu připomíná zápis programu v databázových jazycích typu FOXPRO a DBASE.

Příkazy jazyka popisují algoritmus akce, která se má provést. Příkazy mohou být přitom jednoduché nebo složené. Jednoduché příkazy jsou zapsány celé na jednom řádku a nijak nesouvisí s dalšími řádky programu. Jednoduchým příkazem je například příkaz WRITE. Složené příkazy jsou naopak zapsány na více než jednom řádku programu. Jedná se o příkazy, které rozčlení program do logických celků. Složené příkazy mají vždy definován svůj počátek a konec, přičemž mezi počátkem a koncem příkazu mohou být obsaženy další příkazy. Nejčastěji se jedná o programové cykly, kdy je činnost vymezené části opakována do splnění zadané podmínky. Další možností jsou podmínky, kdy se vnořené příkazy provádí pouze při splnění zadané podmínky.

Složené příkazy mohou být do sebe navzájem vnořovány, nesmí ale docházet ke vzájemným přesahům začátku a konce příkazu:

Správný zápis:

```
IF X > 10
   FOR Y := 10 TO 20
       WRITELN(Y)
   ENDFOR
ENDIF
```

Chybný zápis:

```
IF X > 10
   FOR Y := 10 TO 20
       WRITELN(Y)
   ENDIF      { má být ENDFOR }
ENDFOR        { má být ENDIF  }
```

Příkaz přiřazení (:=)

`Proměnná := Výraz`

Příkaz přiřazení přiděluje proměnné určitou hodnotu. Jméno proměnné je přitom definováno na levé straně příkazu a přiřazovaná hodnota je uvedena na straně pravé.

Proměnná v definici je již dříve definovaná proměnná (příkazem VAR, GLOBAL nebo LOCAL). Výraz je numerický, řetězcový nebo logický výraz který je svým typem totožný přiřazované proměnné.

Viz také:
proměnné, výrazy

Příkaz READ

READ([soubor,] var-name)[;]

Příkaz READ se používá pro vstup údajů z klávesnice, nebo ze souboru. Na rozdíl od standardního Pascalu, je možné vstoupit najednou pouze jednu položku. Pro čtení ze souboru je možné použít pouze textový soubor.

Var-name v definici je proměnná do které se provede načtení dat. Volitelný parametr soubor označuje vstupní soubor (pouze typu TEXT). Pokud není soubor uveden, provádí se čtení ze vstupního okna.

Pro zachování kompatibility se standardním jazykem Psacal je v jazyce definován i příkaz READ, který však provádí stejnou funkci jako příkaz READLN.

Viz také:

proměnné

Příkaz WRITE

WRITE([Soubor,] výraz-1 [:délka][[,] výraz-2[:Délka][[,] výraz-3]])[;]

Příkaz WRITE se používá na zápis hodnot na výstupní zařízení, případně do souboru. Možné je zapisovat najednou i seznam výrazů. Příkaz je velmi podobný standardnímu jazyku Pascal.

Volitelný parametr Soubor v definici je jméno souboru, do kterého bude prováděn výstup. Dosud jsou podporovány pouze textové soubory.

výraz-1 , výraz-2.. jsou výrazy které produkují výstupní hodnotu. V této verzi programu to jsou pouze numerické a řetězcové výrazy. Na rozdíl od Pascalu není možné u numerických výrazů zadat počet desetinných míst.

Viz Také:
příkaz WRITELN, proměnné, výrazy

Příkaz WRITELN

WRITELN([Soubor,] výraz-1 [:délka][[,] výraz-2[:Délka][[,] výraz-3]])[;]

Příkaz WRITELN je velmi podobný příkazu WRITE a používá se na výstup hodnot na výstupní zařízení, případně do souboru. Jediný rozdíl je, že na konci výstupu přidá znak nový řádek, čož u výstupu na obrazovku způsobí, že další výstup bude prováděn na následující řádek.

Volitelný parametr Soubor v definici je jméno souboru, do kterého bude prováděn výstup. Dosud jsou podporovány pouze textové soubory.

výraz-1, výraz-2.. jsou výrazy které produkují výstupní hodnotu. V této verzi programu to jsou pouze numerické a řetězcové výrazy.

Viz Také:
příkaz WRITE, proměnné, výrazy

Příkaz IF .. ELSE .. ENDIF

Příkaz IF slouží pro podmíněné větvení činnosti zpracovávaného programu. Rozdělení činnosti programu se provádí dle vyhodnocení zadané podmínky uvedené za klíčovým slovem IF. Pokud je podmínka splněna, provede se blok příkazů uvedený mezi klíčovými slovy IF a ELSE případě ENDIF. Pokud je uvedeno klíčové slovo ENDIF, provede se blok příkazů uvedených mezi ELSE a ENDIF pouze v případě nesplnění podmínky. Syntaxe příkazu IF je následující:

```
IF podmínka [ THEN ]
   ... příkazy, které se provedou při splnění podmínky
[ ELSE
   ...  příkazy které  se  mají  provést,
        jestliže  není podmínka splněna ]
ENDIF
```

Kde podmínka je logický výraz který může být vyhodnocen hodnotou True nebo False.

Na rozdíl od standardního jazyka Pascal se musí každý příkaz IF ukončit klíčovým slovem ENDIF. V blocích příkazů nelze použít vymezení bloků klíčovými slovy BEGIN a END jako u jazyka Pascal.

Viz také:
proměnné, výrazy

Příkaz FOR .. ENDFOR

Příkaz FOR se používá pro provedení bloku příkazů s přesně definovaným počtem opakování, přičemž počet opakování je znám ihned při definici příkazu.

```
FOR control-variable:=start-value TO|DOWNTO end-value
                                    [STEP step-value]
   ... blok příkazů
ENDFOR
```

Kde control-value je numerická proměnná která bude použita jako řídící proměnná pro smyčku, start-value je počáteční hodnota přidělené řídící proměnné, end-value je konečná hodnota řídící proměnné. Při neuvedení step-value se hodnota řídící proměnné změní v každém cyklu o hodnotu 1. Pokud je zadána hodnota step-value, bude se hodnota řídící proměnné zvyšovat o hodnotu uvedenou ve step-value.

Při použití klíčového slova TO se bude stav řídící proměnné cyklicky zvyšovat. Je-li uvedeno DOWNTO, bude se hodnota řídící proměnné snižovat. Pokud je rozdíl počáteční a koncové hodnoty menší než nula, cyklus nebude vůbec proveden.

Hodnota řídící proměnné se nesmí v cyklu měnit. Hodnoty cyklu jsou vyhodnocovány pouze jednou a to při spuštění příkazu FOR. U příkazů WHILE a REPEAT jsou podmínky vyhodnocovány na začátku, respektive na konci každého cyklu.

Viz také:
výrazy, příkaz REPEAT, příkaz WHILE

Příkaz REPEAT .. UNTIL

Příkaz se používá v situacích, kdy chcete, aby byly příkazy v cyklu provedeny minimálně jednou, protože podmínka opakování je uvedena na konci bloku příkazů.

```
REPEAT
   ... blok příkazů
UNTIL podmínka-ukončení
```

Kde podmínka-ukončení je logický výraz, který je vyhodnocován na konci každého cyklu. Blok příkazů se provádí tak dlouho, dokud má podmínka-ukončení hodnotu True.

Tento příkaz je různý od příkazu WHILE, protože blok příkazů bude proveden vždy alespoň jednou. U příkazu WHILE nemusí být z důvodu kontroly podmínky na začátku cyklu proveden cyklus ani jednou. Na rozdíl od jazyka Pascal nelze používat pro vymezení bloku cyklu klíčová slova BEGIN a END.

Viz Také:
výrazy, příkaz WHILE, příkaz FOR

Příkaz WHILE .. ENDWHILE

Příkaz WHILE umožní vykonávat opakující se programové sekvence příkazů, které jsou prováděny při splnění dané podmínky. Podmínka je definována na počátku příkazu a pokud je splněna, provede se blok příkazů až k ukončujícímu klíčovému slovu ENDWHILE. Na konci smyčky je znovu vyhodnocována podmínka a blok příkazů se opakuje až do okamžiku nesplnění podmínky. Program potom pokračuje příkazy, které jsou definovány za klíčovým slovem ENDWHILE.

Syntaxe příkazu WHILE je:

```
WHILE Podmínka
   ... blok příkazů
ENDWHILE
```

Kde podmínka je logický výraz který může být vyhodnocen hodnotou True nebo False.

Na rozdíl od jazyka Pascal musí být blok příkazů ukončen vždy klíčovým slovem ENDWHILE. Pro vymezení bloku nelze používat klíčová slova BEGIN a END.

Viz Také:
výrazy, příkaz REPEAT, příkaz FOR

Příkaz RETURN

Příkaz RETURN se používá pro předčasné ukončení procedury nebo funkce s případným předáním návratové hodnoty.

```
RETURN [ výraz ] [;]
```

Kde výraz je výraz který definuje návratovou hodnotu funkce, pokud funkce nebo procedura nevrací hodnotu.

Alternativní syntaxe příkazu RETURN je:

```
FUNCTION myfunc(seznam_parametrů) : Návrat-Typ
   [... Nějaký kód]
   Myfunc := Výraz
   [... Nějaký kód]
ENDPROC
```

Alternativní nastavení výstupní hodnoty funkce se provádí před ukončením funkce a to přiřazením hodnoty návratové proměnné. Tato metoda je ekvivalentní klasickému zápisu v jazyce Pascal.

RETURN je příkaz, který není zahrnut ve standardním Pascalu, jeho obdoba se však používá v jazyce Basic, C/C++ a v databázových jazycích.

Viz Také:
výrazy, definice procedur a funkcí

Příkaz SWITCH .. ENDSWITCH

Příkaz SWITCH se používá pro vícenásobné větvení činnosti programu na základě výsledku jedné z voleb vyhodnocení výsledku. Příkaz obsahuje řídící výraz, tzv. selektor, jehož hodnota po porovnání s výkonnými výrazy určuje, která část programu bude vykonána.

Zápis konstrukce příkazu SWITCH .. ENDSWITH má následující syntaxi:

```
SWITCH výraz [ OF ]
   CASE expr1 :
      ... blok příkazů
   ENDCASE
   [ CASE expr2 :
      ... blok příkazů
   ENDCASE...]
   [ ELSE
      ... blok příkazů
   ENDCASE ]
ENDSWITCH
```

Kde výraz je klíčový výraz pro vyhodnocení větvení programu. Pokud se výsledek uvedeného výrazu rovná některému z výrazů uvedených za klíčovým slovem CASE, bude proveden blok příkazů za tímto výrazem. Program bude potom pokračovat po ukončení konstrukce příkazu klíčovým slovem ENDSWITCH. Pokud se výraz nebude rovnat žádnému z porovnávaných výrazů, provede se blok příkazů uvedený mezi klíčovými slovy ELSE a ENDCASE.

Příkazy SWITCH nemohou výt v INTER-PASCALu do sebe vzájemně vnořovány.

Na rozdíl od standardního jazyka Pascal lze porovnávat řídící výraz s dalšími výrazy. To je možné díky tomu že, INTER-PASCAL je interpretovaný jazyk.

Příkaz CONTINUE

Příkaz CONTINUE se používá pro přerušení cyklu WHILE, REPEAT nebo FOR. Příkaz je vyhodnocen jako ENDWHILE, UNTIL nebo ENDFOR a řízení programu je předáno zpět na začátek cyklu. Klíčové slovo je přitom uvedeno za příkazy, které se mají provést vždy, ještě před ukončením (přerušením) cyklu. Pokud se příkaz použije mimo smyčku, je vyvoláno chybové hlášení.

Příklad:

```
FOR x := 1 to 100;
    IF y[x] = 0 then
       CONTINUE
    ENDIF
    Point(x+y[x], z, 1)   {pro y[x] := 0 se neprovede !}
ENDFOR
```

Viz také:
příkaz WHILE, příkaz REPEAT, příkaz FOR

9.0. KNIHOVNY PROCEDUR A FUNKCÍ

Jazyk INTER-PASCAL definuje sadu knihoven a funkcí, které jsou funkčně založeny ba standardní knihovně jazyka Pascal. Obsahují však některé odlišnosti, které vhodně doplňují nebo rozšiřují jazyk INTER-PASACAL.

9.1. SYSTÉMOVÁ KNIHOVNA

Systémová knihovna obsahuje procedury a funkce související s chodem celého programu. Jejich využití je především z příkazového režimu pro ladění programu.

Beep - program pípne přes reproduktor počítače (speaker)
GetChar - vrací znak poslední stisknuté klávesy
GetKey - vrací kód poslední stisknuté klávesy
KeyPressed - testuje stisknutí klávesy
PortInput - čte hodnotu z portu
PortOutput - zapíše hodnotu na port
RunPrg - spustí externí program *.EXE
Quit - ukončí činnost systému

Změna nastavení oken integrovaného prostředí:

CommandForm - nastaví velikost příkazového okna
CommandHide - minimalizuje příkazové okno do ikony
CommandShow - obnoví původní velikost příkazového okna
ConsoleClear - zruší obsah výstupního textového okna
ConsoleForm - nastaví velikost textového výstupního okna
ConsoleHide - minimalizuje textové výstupní okno do ikony
ConsolePrint - vytiskne obsah textového výstupního okna
ConsoleSave - uloží obsah textového výstupního okna do souboru
ConsoleShow - obnoví původní velikost textového výstupního okna
DbfForm - nastaví velikost databázového okna
DbfHide - minimalizuje databázové okno do ikony
DbfShow - obnoví původní velikost databázového okna
FormAllHide - minimalizuje všechna otevřená okna
ChartForm - nastaví velikost okna s grafem
ChartHide - minimalizuje okno s grafem do ikony
ChartShow - obnoví původní velikost okna s grafem
ImageForm - nastaví velikost grafického okna
ImageHide - minimalizuje grafické okno do ikony
ImageShow - obnoví původní velikost grafického okna
MenuInit - inicializuje (nuluje) menu
MenuChange - nastaví položku uživatelského menu
MenuIcon - nastaví obsah ikony uživatelského menu
MenuActive - aktivuje/deaktivuje uživatelské menu
ProgramForm - nastaví velikost okna s programem
ProgramHide - minimalizuje okno s programem do ikony
ProgramShow - obnoví původní velikost okna s programem
SpreadForm - nastaví velikost okna s tabulkou
SpreadHide - minimalizuje okno s tabulkou do ikony
SpreadShow - obnoví původní velikost okna s tabulkou

Prostředky pro ladění programu:

Halt - ukončí činnost programu, přejde do ladícího režimu
Suspend - program přejde do ladícího režimu
LoadVariable - načte hodnoty proměnných ze souboru
SaveVariable - uloží hodnoty deklarovaných proměnných do souboru

Procedura Beep

```
procedure Beep;
```

Procedura pípne přes reproduktor počítače. Vhodné pro upozornění na ukončení výpočtu, uživatelovu chybu a podobně.

Funkce GetChar

```
function GetChar : char;
```

Funkce vrací znak naposledy stisknuté klávesy. Nereaguje na stisk funkčních a řídících kláves.

Pøíklad použití:

```
s := '';
WHILE (ord(s) <> 27)         {dokud není stisknuta klávesa ESC}
   REPEAT UNTIL KeyPressed; {dokud není stisknuta klávesa}
   s := GetChar;            {načte klávesu}
   WRITE(s);                {vypíše znak stisknuté klávesy}
ENDWHILE
```

Funkce GetKey

```
function GetKey : byte;
```

Funkce vrací kód naposledy stisknuté klávesy. Dokáže rozlišit kódy funkčních a řídících kláves. Program proto může reagovat například na stisk kurzorových kláves.

Pøíklad použití:

```
z := 0;
WHILE (z <> 27)       {dokud není stisknuta klávesa ESC}
   REPEAT UNTIL KeyPressed;
   z := GetKey;
   WRITE(chr(z));     {vypíše znak stisknuté klávesy}
ENDWHILE
```

Funkce KeyPressed

```
function KeyPressed : boolean;
```

Funkce vrací logickou hodnotu udávající, zda byla stisknuta libovolná klávesa. Testuje tedy stisk klávesy. Číselný kód stisknuté klávesy zjistíme následně funkcí GetKey, případně znak stisknuté klávesy zjistíme funkcí GetChar. Test stisknuté klávesy se provádí pouze pokud je aktivováno grafické nebo textové výstupní okno. Doporučuje se proto před použitím funkce jedno z uvedených oken aktivovat.

Pøíklad použití:

```
ConsoleShow:          {aktivuje textové výstupní okno}
z := 0;
WHILE (z <> 27)       {dokud není stisknuta klávesa ESC}
   REPEAT UNTIL KeyPressed;
   z := GetKey;
   ...                {zpracování dle kódu stisknuté klávesy}
ENDWHILE
```

Funkce PortInput

```
function PortInput(port: word) : byte;
```

Procedura umožňuje načíst hodnotu ze zadaného portu počítače. Port je možné zadat buď přímo jeho adresou, případně lze využít předdefinované systémové konstanty:

port	hexadecimálně	dekadicky
LPT1	$378	888
LPT1G	$3BC	956
LPT2	$278	632
COM1	$3F8	1016
COM2	$2F8	760
COM3	$3E8	1000
COM4	$2E8	744

Port označený nestandardně LPT1G bývá umístěn na grafické kartě a může mít jinou adresu, než běžný port LPT1.

Příklad použití:

```
p := PortInput(LPT1+1); {načte stavový registr tiskárny}
p := PortInput($278);
```

UPOZORNĚNÍ:

Jakékoliv neodborné použití příkazu může způsobit poškození počítače. Stejně tak i neodborné připojování zařízení na porty počítače.

Procedura PortOutput

```
procedure PortOutput(port: word, hodnota: byte);
```

Procedura umožňuje posílat data uvedená v parametru hodnota na zadaný port počítače. Port je možné zadat buď přímo jeho adresou, případně lze využít předdefinované systémové konstanty:

port	hexadecimálně	dekadicky
LPT1	$378	888
LPT1G	$3BC	956
LPT2	$278	632
COM1	$3F8	1016
COM2	$2F8	760
COM3	$3E8	1000
COM4	$2E8	744

Port označený nestandardně LPT1G bývá umístěn na grafické kartě a může mít jinou adresu, než běžný port LPT1.

Příklad:

```
PortOutput(LPT1, 255);
PortOutput($278,   0);
```

UPOZORNĚNÍ:

Jakékoliv neodborné použití příkazu může způsobit poškození počítače. Stejně tak i neodborné připojování zařízení na porty počítače.

Funkce RunPrg

```
function RunPrg(fil:string, par:string, dir:string, win:byte,
                wait:boolean) : byte
```

Funkce spustí zadaný externí program typu *.EXE, případně program asociovaný se zadaným souborem. Funkci používejte velmi opatrně a s rozvahou. Zvláště spuštění programů určených pro operační systém MS-DOS by mohlo způsobit v některých případech značné problémy. Například program T602.EXE má na některých počítačích problémy s klávesnicí, případně se svým ukončením. Větší problémy by ale mohlo někdy vyvolat spuštění rezidentního programu.
V demoverzi programu KLONDAIK je funkce RunPrg účinná pouze při zadání z příkazového okna. Při zadání z programu nebude externí program v demoverzi spuštěn.

Parametry:
fil - jméno programu (souboru), který má být aktivován
par - parametr předaný spuštěnému programu
dir - pracovní adresář, ve kterém bude program spuštěn
win - definice typu okna, ve kterém bude program spuštěn
wait - požadavek na čekání ukončení programu (true = ano)

Pokud se nachází aplikace, kterou chcete spustit, v jiném, než aktivním adresáři, musíte zadat i cestu k této aplikaci. Zadaný parametr se předá spuštěnému programu. Pokud zadáváte pracovní adresář, ve kterém se má program spustit, musíte mít definovanou cestu ke spouštěnému programu příkazem PATH při inicializaci operačního systému v souboru AUTOEXEC.BAT.
Místo jména programu můžete uvést přímo soubor, který máte ve Windows asociován s programem, který se spustí a zadaný soubor dále zpracuje. Například při zadání jména souboru 'MANUAL.SAM' se spustí textový editor AmiPro, který ihned načte uvedený dokument. V tomto případě ale musíte uvést pracovní adresář, ve kterém se zadaný soubor nachází. Například:

```
RunPrg('MANUAL.SAM', '', 'C:\DOKUMENT\', 1, true);
```

Druh okna, ve kterém bude spuštěná aplikace spuštěna je uveden v parametru win dle následující tabulky:

hodnota	druh okna pro aplikaci
1	okno v aktuální velikosti a poloze
2	maximalizované okno
9	skryté, neviditelné okno

POZOR !

Skryté neviditelné okno je možné použít pouze pro aplikace, které neprovádí žádnou viditelnou činnost a samy se také po provedení zadaných akcí ukončí !

Programy v operačním systému Windows pracují v režimu multitaskingu, to znamená, že může být spuštěno najednou několik programů, mezi kterými se můžete přepínat. Tato vlastnost by mohla v některých případech být na obtíž. Například pokud by jste chtěli spustit sérii programů, jejichž činnost má na sebe navazovat, byly by programy spuštěny teoreticky najednou bez čekání na ukončení předchozího programu. Proto je volání funkce RunPrg doplněno logickým parametrem wait, který udává, zda se má čekat na ukončení spuštěného programu. Při zadání hodnoty true se bude čekat na ukončení programu a až potom se bude pokračovat ve vykonávání další řádky programu. Zadá-li se hodnota false, je program spuštěn a bez čekání na jeho ukončení se pokračuje ve vykonávání další řádky programu. K zamezení různých kolizí však nedoporučujeme tento

způsob používat.

```
RunPrg('PROGRAM.EXE', '', '', 1, true ); {čeká na ukončení}
RunPrg('PROGRAM.EXE', '', '', 1, false); {nečeká na ukončení!}
```

Funkce RunPrg vrací hodnotu udávající úspěšnost spuštění externího programu. Pokud je vrácena hodnota 32, proběhlo vše v pořádku. Jiné hodnoty signalizují výskyt chyby. Některé z možných chyb jsou uvedeny v následující tabulce:

hodnota	způsob ukončení aplikace
0	nedostatek operační paměti
2	zadaný soubor nebyl nalezen
3	neplatná cesta k souboru
10	neplatná verze systému Windows
15	aplikace nepracuje v chráněném módu

Procedura Quit

```
procedure Quit;
```

Pokud je procedura volána z příkazového okna, je okamžitě ukončena činnost systému (program KLONDAIK je ukončen). Jestliže však umístíme volání procedury do programu, dojde k ukončení systému pouze v případě, že byl program spuštěn přímo z Windows kliknutím na ikonu programu vytvořeného v systém KLONDAIK. V takovém případě se nezobrazí vývojové prostředí, ale je ihned spuštěn zadaný program. Je-li v takovém případě volána z programu procedura Quit, dojde ukončení systému.

Procedura Halt

```
Procedure Halt;
```

Procedura ukončí činnost interpretace programu. V okně program je zobrazen řádek, na kterém u ukončení programu došlo. Při dalším spuštění programu bude jeho činnost vykonávána od jeho počátku.

Viz také: Suspend

Procedura Suspend

```
procedure Suspend;
```

Procedura přeruší činnost interpretace programu a přejde do ladícího režimu. V okně program je zobrazen řádek, na kterém u ukončení programu došlo. Při dalším spuštění programu bude jeho činnost vykonávána od následujícího řádku programu. Zadání procedury z příkazového řádku neprovede žádnou akci.

Viz také: Halt

Procedura LoadVariable

```
procedure LoadVariable(f: string);
```

Procedura načte ze souboru definovaného parametrem f obsah proměnných uložených procedurou SaveVariable. Proměnné nesmí být předem deklarovány, budou deklarovány přímo procedurou LoadVariable. Pokud budou proměnné v programu deklarovány, nebude provedena aktualizace hodnot z načteného souboru. Ze souboru nelze načíst obsah proměnných, které byly definovány uživatelskými typy příkazem TYPE .. ENDTYPE.

Vzhledem k velmi nestandardnímu způsobu práce s proměnnými se nedoporučuje používat proceduru v běžných programech. Vhodné použití je však například pro možnost dodatečného zkoumání obsahu proměnných při ladění programu.

Viz také:
SaveVariable

Procedura SaveVariable

```
procedure SaveVariable(f: string);
```

Procedura uloží obsah všech deklarovaných proměnných do souboru. Jméno souboru se zadává v parametru f. Proceduru je vhodné použít pro možnost pozdější analýzy funkce programu. Lze použít pouze pokud nejsou v programu definovány nové typy proměnných příkazem TYPE .. ENDTYPE. Doporučuje se používat pouze pro ladění programu.

Viz také: LoadVariable

Procedura CommandForm

```
procedure CommandForm(Top, Left, Width, Height :integer);
```

Změní velikost a umístění příkazového okna na ploše systému KLONDAIK. Parametry Left a Top procedury určují levý horní roh v bodech, parametr Width určuje šířku okna a parametr Height zadává výšku okna v bodech.

Viz také:
CommandHide, CommandShow

Procedura CommandHide

```
procedure CommandHide;
```

Procedura minimalizuje do tvaru ikony příkazové okno. Proceduru je vhodné použít na začátku programu pro odstranění nepoužívaných oken a tím ke zpřehlednění obrazovky.

Procedura CommandShow

```
procedure CommandShow;
```

Procedura obnoví velikost příkazového okna do podoby před jeho minimalizací do ikony.

Procedura ConsoleClear

```
procedure ConsoleClear;
```

Procedura vymaže obsah textového výstupního okna. Proceduru je vhodné použít na začátku programu pro odstranění výstupu předchozího programu.

Procedura ConsoleForm

```
procedure ConsoleForm(Top, Left, Width, Height :integer);
```

Změní velikost a umístění textového výstupního okna na ploše systému KLONDAIK. Parametry Left a Top procedury určují levý horní roh v bodech, parametr Width určuje šířku okna a parametr Height zadává výšku okna v bodech.

Procedura ConsoleHide

```
procedure ConsoleHide;
```

Procedura minimalizuje do tvaru ikony výstupní textové okno. Proceduru je vhodné použít na začátku programu pro odstranění nepoužívaných oken a tím ke zpřehlednění obrazovky.

Procedura ConsolePrint

```
procedure ConsolePrint;
```

Procedura vytiskne obsah textového výstupního okna na aktuálně nastavenou tiskárnu.

Funkce ConsoleSave

```
procedure ConsoleSave(files: string);
```

Procedura uloží obsah textového výstupního okna na disk do souboru definovaného parametrem files. Jestliže soubor zadaného jména již existuje, bude přepsán.

Procedura ConsoleShow

```
procedure ConsoleShow;
```

Procedura obnoví velikost výstupního textového okna do podoby před jeho minimalizací do ikony.

Procedura DbfForm

```
procedure DbfForm(Top, Left, Width, Height :integer);
```

Změní velikost a umístění databázového okna na ploše systému KLONDAIK. Parametry Left a Top procedury určují levý horní roh v bodech, parametr Width určuje šířku okna a parametr Height zadává výšku okna v bodech.

Viz také:
DbfHide, DbfShow

Procedura DbfHide

```
procedure DbfHide;
```

Procedura minimalizuje do tvaru ikony databázové okno. Proceduru je vhodné použít na začátku programu pro odstranění nepoužívaných oken a tím ke zpřehlednění obrazovky.

Procedura DbfShow

```
procedure DbfShow;
```

Procedura obnoví velikost databázového okna do podoby před jeho minimalizací do ikony.

Procedura ChartForm

procedure ChartForm(Top, Left, Width, Height :integer);

Změní velikost a umístění okna s grafem na ploše systému KLONDAIK. Parametry Left a Top procedury určují levý horní roh v bodech, parametr Width určuje šířku okna a parametr Height zadává výšku okna v bodech.

Viz také:
ChartHide, ChartShow

Procedura FormAllHide

```
procedure FormAllHide;
```

Slouží k minimalizaci všech otevřených oken systému KLONDAIK. Je vhodné použít nejlépe na začátku každého spuštěného programu, ve kterém se potom dále nastaví velikost a poloha dále používaných oken.

Viz také:
CommandHide, ConsoleHide, DbfHide, ChartHide, ImageHide, ProgramHide, SpreadHide

Procedura ImageForm

```
procedure ImageForm(Top, Left, Width, Height :integer);
```

Změní velikost a umístění grafického výstupního okna na ploše systému KLONDAIK. Parametry Left a Top procedury určují levý horní roh v bodech, parametr Width určuje šířku okna a parametr Height zadává výšku okna v bodech.

Viz také:
ImageHide, ImageShow

Procedura ImageHide

```
procedure ImageHide;
```

Procedura minimalizuje do tvaru ikony grafické výstupní okno. Proceduru je vhodné použít na začátku programu pro odstranění nepoužívaných oken a tím ke zpřehlednění obrazovky.

Procedura ImageShow

```
procedure ImageShow;
```

Procedura obnoví velikost grafického výstupního okna do podoby před jeho minimalizací do ikony.

Procedura ProgramForm

```
procedure ProgramForm(Top, Left, Width, Height :integer);
```

Změní velikost a umístění okna s programem na ploše systému KLONDAIK. Parametry Left a Top procedury určují levý horní roh v bodech, parametr Width určuje šířku okna a parametr Height zadává výšku okna v bodech.

Viz také:
ProgramHide, ProgramShow

Procedura ProgramHide

```
procedure ProgramHide;
```

Procedura minimalizuje do tvaru ikony okno s programem. Proceduru je vhodné použít na začátku programu pro odstranění nepoužívaných oken a tím ke zpřehlednění obrazovky.

Procedura ProgramShow

```
procedure ProgramShow;
```

Procedura obnoví velikost okna s programem do podoby před jeho minimalizací do ikony.

Procedura SpreadForm

```
procedure SpreadForm(Top, Left, Width, Height :integer);
```

Změní velikost a umístění okna s tabulkou na ploše systému KLONDAIK. Parametry Left a Top procedury určují levý horní roh v bodech, parametr Width určuje šířku okna a parametr Height zadává výšku okna v bodech.

Viz také:
SpreadHide, SpreadShow

Procedura MenuInit

```
procedure MenuInit;
```

Procedura nuluje uživatelské menu a připravuje jej na novou definici. Po inicializaci obsahuje menu pouze jednu položku 'Konec programu', která se používá pro ukončení uživatelského menu s návratem do systémového menu. Uživatelské ikony toolbaru jsou po inicializaci neviditelné. Po inicializaci menu se následně volanými procedurami MenuChange nastaví položky menu. Procedurou MenuIcon se nastaví uživatelské ikony v toolbaru. Uživatelské menu se potom aktivuje procedurou MenuActive.

Uživatelské menu může obsahovat maximálně šest hlavních položek, z nichž první může mít deset podřízených voleb (položek), ostatní potom mohou obsahovat pouze pět podřízených voleb. Dále je možné použít pro volání programů deset uživatelských ikon v toolbaru. Celkem je tedy možné volat z uživatelského menu 45 uživatelem zadaných programů.

Viz také:
MenuChange, MenuIcon, MenuActive

Procedura MenuChange

```
procedure MenuChange(x, y: byte, text, prg, hlp: string);
```

Nastavuje položku uživatelského menu. Úroveň položky menu se zadává parametry x a y. Parametr x udává číslo hlavní položky menu a může být v rozsahu 1 až 6. V parametru y se zadává číslo podřízené položky menu a může být v rozsahu 0 až 10 pro první hlavní položku menu a 0 až 5 pro ostatní položky menu. Pokud je tedy parametr y nulový, zadává se stále viditelná položka v hlavního menu.

V parametru text se zadává text položky menu. Jméno volaného programu se zadává v parametru prg, parametr hlp obsahuje nápovědu k položce menu, která se zobrazuje ve stavovém řádku.

Pokud nastavíte opakovaně jednu položku menu s různými parametry volaných programů, bude se při použití menu brát v úvahu vždy poslední nastavení. To je možné použít například pro dynamickou změnu obsahu uživatelského menu. Podrobněji viz kurz používání programu.

Viz také:
MenuInit, MenuIcon, MenuActive

Procedura MenuIcon

```
procedure MenuIcon(x: byte, ico, prg, hlp: string);
```

Nastaví ikonu v uživatelském toolbaru. K dispozici je deset ikon, číslo ikony se zadává v parametru x. Název souboru s ikonou typu BMP se zadává v parametru ico. Jméno programu, který se spustí se uvádí v parametru prg. Je přitom možné uvést běžné programy (*.IPS) nebo již zakódované programu (*.IPX). V parametru hlp je možné uvést nápovědu, která se zobrazí po najetí kurzoru nad ikonu.

Jako ikona se používá bitmapa velikosti 16x16 bodů uložená v souboru typu *.BMP. Pokud budete chtít ponechat v řadě ikon mezeru, vynechejte definici jedné ikony. Tím vznikne mezera o šířce jedné ikony.

Viz také:
MenuInit, MenuChange, MenuActive

Procedura MenuActive

```
procedure MenuActive(act: boolean);
```

Procedura podle stavu parametru act buď aktivuje nebo deaktivuje uživatelské menu. Pokud se zadá jako parametr act hodnota True, zobrazí se místo menu systému KLONDAIK nadefinované uživatelské menu. Současně s tím se stane nepřístupné okno s programem, příkazové okno, okno s texty a okno sledování proměnných. Původní stav menu systému KLONDAIK a používaných oken se vrátí buď opětovným voláním procedury MenuActive s parametrem False, nebo ukončením uživatelského menu z volby 'Konec programu'.

Viz také:
MenuInit, MenuChange, MenuIcon

9.2. MATEMATICKÁ KNIHOVNA

Matematická knihovna umožňuje provádění základních matematických výpočtů s čísly. Obsahuje také základní trigonometrické funkce.

Abs - vrací absolutní hodnotu čísla
ArcTan - vrací arkustangens čísla
Cos - vrací kosinus čísla
Dec - zmenšuje hodnotu proměnné
exp - vrací exponent čísla
Inc - zvyšuje hodnotu proměnné
ln - vrací přirozený logaritmus argumentu
Low - vrací nejnižší možnou hodnotu indexu pole
Max - vrací větší hodnotu ze dvou zadaných čísel
Min - vrací menší hodnotu ze dvou zadaných čísel
High - vrací nejvyšší možnou hodnotu indexu pole
PI - vrací hodnotu ludolfova čísla
Random - vrací náhodné číslo
Round - zaokrouhluje číslo na celočíselnou hodnotu
Sin - vrací hodnotu sinus čísla
Sgr - vrací druhou mocninu čísla
Sqrt - vrací druhou odmocninu čísla
Trunc - odřízne desetinnou část reálného čísla

Funkce abs

```
function abs(r: real) : real;
```

Vrací absolutní hodnotu čísla r.

Funkce Arctan

```
function arctan(r: real) : real;
```

Funkce vrací arkustangens čísla r.

Funkce cos

```
function cos(r: real ) : real;
```

Vrací kosinus čísla r.

Procedura Dec

```
procedure Dec(var v[, n: real]);
```

Procedura zmenšuje hodnotu proměnné v, která musí být numerického typu. Pokud je definován volitelný parametr n, bude provedeno snížení hodnoty o velikost parametru n, jinak se hodnota v sníží o jedničku.

Funkce exp

```
function exp(r: real) : real;
```

Vrací exponent čísla r.

Procedura Inc

```
Procedura Inc(var v[, n: real]);
```

Zvyšuje hodnotu proměnné v, která musí být numerického typu. Pokud je definován volitelný parametr n, bude provedeno zvýšení hodnoty proměnné o velikost parametru n, jinak se hodnota v zvýší o jedničku.

Funkce ln

```
function ln(r: real) : real;
```

Vrací přirozený logaritmus argumentu.

Funkce Low

```
function Low(var arrayVar: array) : integer;
```

Funkce vrací nejnižší možnou hodnotu indexu pole. Funkce se používá většinou pro zjištění rozsahu otevřeného pole v rámci procedury.

Příklad:

```
FOR i := Low(vector) TO High(vector)
   WRITELN(vector[i]);
ENDFOR
```

Funkce High

```
function High(var arrayVar : array) : integer;
```

Funkce vrací nejvyšší možnou hodnotu indexu pole. Funkce se používá většinou pro zjištění rozsahu otevřeného pole v rámci procedury.

Příklad:

```
FOR i := Low(vector) TO High(vector)
   WRITELN(vector[i]);
ENDFOR
```

Funkce Max

```
function Max(r1: real, r2: real) : real;
```

Funkce porovná hodnoty vstupujících čísel a vrací hodnotu většího čísla.

Viz také: Min

Funkce Min

```
function Min(r1: real, r2: real) : real;
```

Funkce porovná hodnoty vstupujících čísel a vrací hodnotu menšího čísla.

Viz také: Max

Funkce Pi

```
Const Pi = 3.14;
```

Funkce vrací hodnotu Ludolfova čísla 3.1415926535897932385.

Funkce Random

```
function random(n: longint) : longint;
```

Funkce vrací náhodné číslo z intervalu 0 až n.

Funkce Round

```
function round(r: real) : longint;
```

Funkce zaokrouhluje parametr reálného typu na celočíselnou hodnotu. Výsledek funkce je celočíselná hodnota, která je nejbližší k hodnotě parametru. Čísla, která jsou uprostřed dvou celých čísel se zaokrouhlují k číslu, které má větší absolutní hodnotu.

Funkce Sin

`function sin(r: real) : real;`

Vrací hodnotu sinus čísla r.

Funkce Sqr

```
function sqr(r: real) : real;
```

Vrací druhou mocninu čísla r.

Viz také: Sqrt

Funkce Sqrt

```
function sqrt(r: real) : real;
```

Vrací druhou odmocninu z čísla r.

Viz také: Sqr

Funkce Trunc

```
function trunc(r: real) : longint;
```

Provádí odříznutí desetinné části reálného parametru a vrací celočíselnou hodnotu.

9.3. KNIHOVNA PRO ZPRACOVÁNÍ ŘETĚZCŮ

Knihovna obsahuje procedury a funkce pro práci s řetězci. Mimo zpracování podřetězců umožňuje převod čísel na řetězec a zpět.

Copy	- vrací zadanou část řetězce
Day	- vrací jméno dne v týdnu
Delete	- vyjme část znaků z řetězce
Chr	- na základě čísla z ASCII tabulky vrací příslušný znak
Insert	- vloží podřetězec do řetězce
IntToStr	- převádí číslo do tvaru řetězce
Length	- vrací délku řetězce
LowerCase	- konvertuje řetězec na malá písmena
Month	- vrací jméno měsíce v roce
Ord	- vrací ASCII hodnotu znaku
Pos	- vyhledá podřetězec v řetězci
Space	- vrací prázdný řetězec zadané délky
Str	- provádí konverzi čísla na řetězec
StrToInt	- převádí řetězec do číselné hodnoty
StrToReal	- převádí řetězec do číselné hodnoty typu real
Trim	- uřízne z řetězce počáteční a koncové mezery
TrimLeft	- uřízne z řetězce počáteční mezery
TrimRight	- uřízne z řetězce koncové mezery
UpperCase	- konvertuje řetězec na velká písmena
Val	- provádí konverzi řetězce do numerické hodnoty

Funkce Ord

```
function ord(c: char) : byte;
```

Funkce vrací originální hodnotu argumentu. Jedná se o ASCII hodnotu odpovídající znaku.

Funkce Chr

```
function chr(b: byte) : char;
```

Funkce vrací znak, který je reprezentován zadanou celočíselnou hodnotou (dekadické vyjádření ASCII znaku).

Funkce Copy

function Copy(s: string, i: byte, l: byte) : string;

Funkce vrací podřetězec z řetězce s. Parametr i udává počáteční index v původním řetězci a parametr l uvádí počet znaků přenesených do výsledného řetězce počínajíc i-tým znakem. Je-li hodnota parametru i větší, než délka řetězce s, je vrácen prázdný řetězec. Je-li hodnota parametru l větší, než zbývající délka řetězce, je přenesen pouze zbytek řetězce s.

Funkce Length

```
function Length(s : string) : byte;
```

Funkce vrací délku řetězce s.

Funkce Insert

```
function Insert(s: string, var d: string, i: integer);
```

Funkce vkládá podřetězec s do řetězce daného parametrem d. Hodnota zadaná parametrem i udává počáteční hodnotu indexu v řetězci s, kam budce vložen řetězec s.

Procedura Delete

```
procedure Delete(var s: string, i: integer, c: integer);
```

Procedura zruší c znaků z podřetězce s od pozice i.

Funkce Pos

```
function Pos(subs: string, s: string) : integer;
```

Funkce vyhledá podřetězec subs v řetězci s. Je-li subs v řetězci s nalezen, vrací funkce hodnotu indexu prvního znaku v řetězci s, kde byl subs nalezen. Neobsahuje-li řetězec s zadaný podřetězec, je vrácena hodnota 0.

Funkce Val

```
function Val(s: string, var r: real) : integer;
```

Funkce provádí konverzi řetězce s do odpovídající numerické hodnoty. Pokud je vrácena nenulová hodnota parametru r, obsahuje index (pořadí) znaku, u kterého došlo k chybě při konverzi.

Procedura Str

```
procedure Str(r: [:width [:decimal]], var s: string);
```

Procedura provádí konverzi numerické hodnoty r na odpovídající řetězcovou reprezentaci ve formě znakové proměnné. Volitelně lze zadat parametr width, kterým se specifikuje celkový počet znaků, který výsledný řetězec zaujme, případně parametr decimal, kterým se specifikuje počet desetinných míst.

Funkce IntToStr

```
function IntToStr(i: longint) : string;
```

Funkce převádí číslo v proměnné i do tvaru řetězce.

Funkce StrToInt

```
function StrToInt(s: string) : longint;
```

Převádí řetězec do číselné hodnoty typu longint.

Funkce StrToReal

```
function StrToReal(s: string) : real;
```

Převádí řetězec do číselné hodnoty typu real. Na rozdíl od funkce Val dokáže převést i obsah řetězce obsahujícího nenumerické znaky, které jsou od svého prvého výskytu až do konce řetězce ignorovány.

Funkce Day

```
function Day(d: integer) : string;
```

Funkce vrací na základě pořadového čísla dne v týdnu jeho jméno jako řetězec.

Viz také: Month

Funkce LowerCase

```
function LowerCase(s : string) : string;
```

Převede všechny znaky zadaného řetězce na malá písmena. Pracuje včetně českých znaků (pokud máte správně nastaven systém Windows).

Viz také: UpperCase

Funkce Month

```
function Month(m: integer) : string;
```

Funkce vrací na základě pořadového čísla měsíce v roce jeho jméno jako řetězec.

Viz také: Day

Funkce Space

```
function Space(d : integer) : string;
```

Funkce vrací řetězec sestavený ze zadaného počtu mezer (prázdných znaků).

Funkce Trim

```
function Trim(s : string) : string;
```

Funkce uřízne ze zadaného řetězce počáteční a koncové mezery.

Viz také: TrimLeft, TrimRight

Funkce TrimLeft

```
function TrimLeft(s : string) : string;
```

Funkce uřízne ze zadaného řetězce počáteční mezery.

Viz také: Trim, TrimRight

Funkce TrimRight

```
function TrimRight(s : string) : string;
```

Funkce uřízne ze zadaného řetězce koncové mezery.

Viz také: Trim, TrimLeft

Funkce UpperCase

```
function UpperCase(s : string) : string;
```

Převede všechny znaky zadaného řetězce na velká písmena. Pracuje včetně českých znaků (pokud máte správně nastaven systém Windows).

Viz také: LowerCase

9.4. KNIHOVNA PRO PRÁCI S TXT SOUBORY

Jazyk INTER-PASCAL je možné použít ke zpracování textových souborů, které se vyznačují členěním na řádky nestejné délky. Textové soubory je možné zpracovávat pouze sekvenčně, což znamená postupně řádek po řádku. Pokud požadujete použití přímého přístupu pomocí vyhledávacích klíčů, použijte zpracování záznamů v tabulkách.

Knihovna procedur a funkcí slouží k vytvoření, zápisu a čtení souborů typu text.

Assign - definuje konkrétní textový soubor
Reset - otevírá existující soubor pro čtení
Close - uzavírá otevřený soubor
Append - otevírá soubor pro přidání nových záznamů
Rewrite - zakládá nový soubor a otevírá jej pro zápis
Readln - čtení ze souboru
Writeln - zápis do souboru
Eof - příznak konce souboru při čtení

Příklad zpracování textových souborů:

```
PROCEDURE main
   VAR
      f: text = "test.dta" { definuje textový soubor }
      s: string
   ENDVAR

   IF (Reset(f) = 0)        { otevře textový soubor }
      WHILE (NOT EOF(f))    { dokud není konec souboru }
         READLN(f,s)        { čte řádky souboru }
         WRITELN(s)         { vypíše řádek na obrazovku }
      ENDWHILE              { konec cyklu }
      Close(f)              { uzavře soubor }
   ENDIF
ENDPROC
```

Procedura Assign

```
procedure Assign(t: Text, s: string);
```

Procedura přiřazuje proměnné typu soubor jméno konkrétního souboru. Parametr t je typu soubor, parametr s obsahuje jméno souboru.

Deklarace souboru je možná nejen pomocí procedury assign, ale také přímo při deklaraci souboru. Následující dva zápisy deklarace souboru jsou proto ekvivalentní :

```
{varianta a:}
VAR
   T: Text;
ENDVAR
assign(T,"Myfile.txt");

{varianta b:}
VAR
   T: Tex = "Myfile.txt";
ENDVAR
```

Viz také:
Reset, Close, Append, Rewrite, Eof

Funkce Reset

```
function Reset(t : Text) : byte;
```

Funkce otevírá existující soubor pro čtení. Parametr t musí být spojen se jménem externího souboru pomocí procedury assign. Byl-li soubor již otevřen, je nejprve uzavřen a poté znovu otevřen. Funkce vrací informaci o úspěšnosti provedení akce. V případě úspěšnosti vrací nulovou hodnotu.

Viz také:
Assign, Close, Append, Rewrite, Eof

Procedura Close

```
function Close(t : Text) : byte;
```

Funkce uzavírá otevřený soubor s návratem chybového kódu. Pokud proběhla akce úspěšně, je vrácena nulová hodnota.

Viz také:
Assign, Reset, Append, Rewrite, Eof

Funkce Append

```
Funkce Append(t : Text) : byte;
```

Otvírá existující soubor t (typu TEXT) pro zápis a nastavuje příští pozici souboru na konec souboru. Funkce vrací chybový kód. Pokud vrátí 0, nevyskytla se žádná chyba.

Viz také:
Assign, Reset, Close, Rewrite, Eof

Funkce Rewrite

```
function Rewrite(t : Text) : byte;
```

Funkce otevírá existující soubor pro zápis a nastavuje pozici souboru na konec souboru. Není-li nalezen specifikovaný soubor, nebo není možné do něj zapisovat, vrací funkce nenulovou hodnotu. Je-li použita funkce na již otevřený soubor, je tento nejprve uzavřen a poté znovu otevřen.

Viz také:
Assign, Reset, Close, Append, Eof

Funkce Eof

```
function Eof(t : Text) : Boolean;
```

Funkce nabývá hodnotu TRUE, pokud bylo dosaženo konce souboru.

Viz také:
Assign, Reset, Close, Append, Rewrite

9.5. KNIHOVNA PRO ZPRACOVÁNÍ ZÁZNAMŮ

Jazyk INTER-PASCAL umožňuje pracovat přímo pouze se soubory typu text, které je možné zpracovávat pouze sekvenčně. Proto byl jazyk doplněn několika procedurami a funkcemi, které umožňují práci se záznamy seskupenými do tabulek. Záznam je (viz definice typů) datová struktura položek. Seskupením položek do tabulky je možné k jednotlivým záznamům přistupovat pomocí klíčů. Celou tabulku je možné také uložit do souboru na disk, případně načíst z disku do paměti.

Seznam procedur a funkcí:

WriteRecord	- zapíše nový záznam do tabulky
ReadRecord	- čte záznam z tabulky
SetKeysFromRecord	- nastaví vyhledávací klíč na zadanou hodnotu
RecordExists	- vyhledává v tabulce požadovaný klíč
DeleteRecord	- zruší záznam z tabulky
ForEachRecord	- provádí zadanou proceduru s každým záznamem
LoadTable	- načte tabulku z disku
SaveTable	- uloží tabulku na disk

Kompletní příklad:

```
TYPE
   customerrecord = RECORD              { definice záznamu }
      KEY firma : string                { definice klíče }
          mesto : string
   ENDRECORD
ENDTYPE

VAR
   customer : customerRecord
   hledej   : customerRecord
   custtable: table of customerRecord   { definice tabulky }
ENDVAR

PROCEDURE printCustomer
   WRITELN(hledej.firma, ', ', hledej.mesto)
ENDPROC

PROCEDURE main
   customer.firma := 'OZOGAN'
   customer.mesto := 'Liberec'
   writerecord(custtable, customer) {zápis záznamu do tabulky}
   customer.firma := 'SH PLUS'
   customer.mesto := 'Liberec'
   writerecord(custtable, customer) {zápis záznamu do tabulky}

   hledej.firma := 'SH PLUS'
   setKeysFromRecord(custtable, hledej) { nastavení klíče }
   IF (readRecord(custtable, hledej)    { vyhledání klíče }
      printCustomer
   ELSE
      WRITELN('nenalezeno')
   ENDIF
ENDPROC
```

Procedura WriteRecord

```
procedure writeRecord(Table: tableVariable, theRecord);
```

Zapíše nový záznam do tabulky. Pokud záznam se stejnými klíči již existuje, bude tento záznam zrušen a nahrazen novým. Záznam se zapíše na místo dle pořadí podle definovaného klíče.

Příklad:

```
customer.firma := 'OZOGAN'
customer.mesto := 'Liberec'
writerecord(custtable, customer)
```

Funkce ReadRecord

```
function readRecord(Table: TableVariable; var targetRecord:
recordVariable) : Boolean;
```

Čte záznam z tabulky po předchozím vyhledání požadovaného záznamu procedurou setKeysFromRecord. Pokud byl požadovaný záznam nalezen a správně načten, vrací funkce hodnotu True, při neúspěšném provedení vrátí hodnotu False.

Příklad:

```
hledej.firma := 'abc'                    {zadání klíče}
setKeysFromRecord(custtable, hledej)     {nastavení klíče}
IF (readRecord(custtable, hledej)        {čtení záznamu dle klíče}
   ...  akce které se provedou v případě úspěšného čtení
ELSE
   ...  akce, které se provedou pokud bylo čtení neúspěšné
ENDIF
```

Procedura SetKeysFromRecord

```
procedure setKeysFromRecord(table: TableVariable; Record:
RecordVariable);
```

Procedura nastaví vyhledávací klíč na zadanou hodnotu použitelnou pro vyhledání požadovaného záznamu. Neprovádí vyhledání. To je realizováno následnými příkazy. Proceduru použijte například pokud chcete vymazat určitý záznam.

Příklad:

```
hledej.firma := 'abc'                    {zadání klíče}
setKeysFromRecord(custtable, hledej)     {nastavení klíče}
IF (readRecord(custtable, hledej)        {hledání dle klíče}
   ....                                  {akce při nenalezení}
ENDIF
```

Funkce RecordExists

```
function recordExists(Table : TableVariable) : Boolean;
```

Prohledává tabulku a vyhledá záznam zadaný procedurou setKeysFromRecord. Pokud bylo hledání úspěšné, vrací funkce hodnotu True, při neúspěšném provedení vrací hodnotu False.

Příklad:

```
hledej.firma := 'abc'                    {zadání klíče}
setKeysFromRecord(custtable, hledej)     {nastavení klíče}
IF (recordExist(custTable)               {hledání dle klíče}
  ... akce, které se provedou při úspěšném hledání
ENDIF
```

Funkce DeleteRecord

```
function deleteRecord(theTable : TableVariable) : Boolean;
```

Zruší záznam, který byl nastaven vyhledáním procedurou SetKeysFromRecord. Funkce vrací hodnotu True, pokud byl záznam nalezen a zrušen, jinak vrací hodnotu False.

Příklad:

```
hledej.firma := 'abc'                  {zadání klíče}
setKeysFromRecord(custtable, hledej)   {nastavení klíče}
deleteRecord(custTable)              {výmaz, pokud klíč nalezen}
```

Procedura ForEachRecord

```
procedure forEachRecord(Table: tableVariable; TargetRecord:
                        recordVariable; ProcName: string);
```

Procedura aktivuje zadanou proceduru v parametru ProcName pro každý záznam v tabulce. Záznamy jsou zpracovávány v pořadí dle řazení klíčových položek v tabulce. Po vyvolání procedury ProcName obsahuje záznam TargetRecord obsah položek zpracovávaného záznamu z tabulky zadané parametrem Table.

Příklad:

```
forEachRecord(CustTable, customer, 'PrintCustomer');

PROCEDURE PrintCustomer
   WRITELN(customer.firma, ', ', customer.mesto)
ENDPROC
```

Funkce LoadTable

```
function LoadTable(Table:jménoTabulky; Soubor:string):Boolean;
```

Funkce LoadTable se používá pro načtení existující tabulky z diskového souboru do paměti. Soubor musí být přitom vytvořen funkcí SaveTable. Funkce vrací v případě úspěšného provedení hodnotu True, při neúspěchu vrací hodnotu False.

Příklad:

```
LoadTable(CustTable, 'customer.dta');
```

Procedura SaveTable

```
procedure saveTable(Table: jménoTabulky; Soubor: string);
```

Procedura uloží tabulku zadanou parametrem Table do souboru na disk s názvem zadaným v parametru Soubor.

Příklad:

```
saveTable(CustTable, 'customer.dta');
```

9.6. KNIHOVNA PRO PRÁCI SE SOUBORY

Knihovna obsahuje základní operace se soubory a adresáři na disku. Většinu těchto funkcí je nutné používat velmi opatrně, protože v případě výmazu souboru můžete přijít o důležitá data.

FileCopy	- zkopíruje soubor do nového adresáře
FileDelete	- zruší zadaný soubor
FileExists	- zjišťuje existenci souboru
FileRename	- přejmenuje soubor
FileSize	- zjišťuje délku souboru
DiskFree	- vrací volné místo na disku v bajtech
DiskSize	- vrací celkovou velikost disku v bajtech
ChDir	- změní nastavený pracovní adresář
GetDir	- vrací jméno aktuálního adresáře
MkDir	- založí nový adresář
RmDir	- zruší prázdný adresář
ForEachFile	- provádí akci se všemi soubory v adresáři

Funkce FileCopy

```
function FileCopy(f1: string, f2: string) : boolean;
```

Funkce kopíruje soubor zadaný v parametru f1 na místo zadané parametrem f2. Pokud proběhne kopírování v pořádku, vrátí funkce hodnotu True, v opačném případě vrací hodnotu False.

Poznámka:
Při práci se soubory na disku není nutné pracovat s celým označením souboru, to je jmeno_disku:\ adresář\soubor. Pokud se bude pracovat pouze se soubory v adresáři, ve kterém je uložen prováděný program, je možné uvádět pouze jméno souboru bez označení disku a adresáře. Obdobně není nutné uvádět jméno disku, pokud je požadovaný disk aktuální a bude uvedena kompletní cesta k souboru (adresář).

Viz také:
FileDelete, FileRename, FileExists, FileSize

Funkce FileDelete

```
function FileDelete(f1: string) : boolean;
```

Funkce zruší soubor zadaný v parametru f1. Pokud proběhne výmaz v pořádku, vrátí funkce hodnotu True, v opačném případě vrací hodnotu False.

Poznámka:
Při práci se soubory na disku není nutné pracovat s celým označením souboru, to je jmeno_disku:\ adresář\soubor. Pokud se bude pracovat pouze se soubory v adresáři, ve kterém je uložen prováděný program, je možné uvádět pouze jméno souboru bez označení disku a adresáře. Obdobně není nutné uvádět jméno disku, pokud je požadovaný disk aktuální a bude uvedena kompletní cesta k souboru (adresář).

Viz také:
FileCopy, FileRename, FileExists, FileSize

Funkce FileExists

```
function FileExists(f1: string) : boolean;
```

Funkce zjišťuje, zda existuje soubor zadaný v parametru f1. Pokud ano vrátí funkce hodnotu True, v opačném případě vrací hodnotu False.

Poznámka:
Při práci se soubory na disku není nutné pracovat s celým označením souboru, to je jmeno_disku:\ adresář\soubor. Pokud se bude pracovat pouze se soubory v adresáři, ve kterém je uložen prováděný program, je možné uvádět pouze jméno souboru bez označení disku a adresáře. Obdobně není nutné uvádět jméno disku, pokud je požadovaný disk aktuální a bude uvedena kompletní cesta k souboru (adresář).

Viz také:
FileCopy, FileDelete, FileRename, FileSize

Funkce FileRename

```
function FileRename(f1: string, f2: string) : boolean;
```

Funkce kopíruje soubor zadaný v parametru f1 na místo zadané parametrem f2. Pokud proběhne kopírování v pořádku, vrátí funkce hodnotu True, v opačném případě vrací hodnotu False.

Poznámka:
Při práci se soubory na disku není nutné pracovat s celým označením souboru, to je jmeno_disku:\ adresář\soubor. Pokud se bude pracovat pouze se soubory v adresáři, ve kterém je uložen prováděný program, je možné uvádět pouze jméno souboru bez označení disku a adresáře. Obdobně není nutné uvádět jméno disku, pokud je požadovaný disk aktuální a bude uvedena kompletní cesta k souboru (adresář).

Viz také:
FileCopy, FileDelete, FileExists, FileSize

Funkce FileSize

```
function FileSize(f1: string) : longint;
```

Funkce zjišťuje velikost souboru zadaného parametrem f1. Výsledek vrací v bajtech. Pokud soubor neexistuje, případně se jej nepodařilo otevřít, vrací funkce hodnotu -1.

Poznámka:
Při práci se soubory na disku není nutné pracovat s celým označením souboru, to je jmeno_disku:\ adresář\soubor. Pokud se bude pracovat pouze se soubory v adresáři, ve kterém je uložen prováděný program, je možné uvádět pouze jméno souboru bez označení disku a adresáře. Obdobně není nutné uvádět jméno disku, pokud je požadovaný disk aktuální a bude uvedena kompletní cesta k souboru (adresář).

Viz také:
FileCopy, FileDelete, FileRename, FileExists

Funkce DiskFree

```
function DiskFree(d: byte) : longint;
```

Funkce zjišťuje velikost neobsazeného místa na zadaném disku. Výsledek vrací v bajtech. Pokud je disk nepřístupný, vrací funkce hodnotu -1. Požadovaný disk se zadává číslem počínaje od jedničky pro disk A. (A=1, B=2, C=2, ...).

Příklad:

```
WRITE('Volné místo na disku C: ')
WRITE(IntToStr(DiskFree(3)/1024/1024), ' Mb')
```

Viz také: DiskSize

Funkce DiskSize

```
function DiskSize(d: byte) : longint;
```

Funkce zjišťuje velikost zadaného disku. Výsledek vrací funkce v bajtech. Pokud je disk nepřístupný, vrací funkce hodnotu -1. Požadovaný disk se zadává číslem počínaje od jedničky pro disk A. (A=1, B=2, C=2, ...). To lze výhodně použít například pro kontrolu připravenosti diskety v mechanice, případně pro zjištění počtu připojených disků (C, D, E, ...).

Příklad:

```
WRITE('Velikost disku C: ')
WRITE(IntToStr(DiskSize(3)/1024/1024), ' Mb')
```

Viz také: DiskFree

Funkce ChDir

```
function ChDir(s: string) : byte;
```

Funkce změní současný pracovní adresář. Parametr s obsahuje specifikaci cesty včetně případné disketové jednotky. Pokud byla změna adresáře úspěšná vrací funkce nulu, jinak chybový kód.

Viz také: GetDir, MkDir

Funkce GetDir

```
function GetDir(d: byte) : string
```

Funkce vrací jméno aktuálního adresáře na libovolném disku. V numerickém parametru d se zadává disková jednotka. Pokud je hodnota nulová, předpokládá se aktuální jednotka. V jiných případech se udává pořadí jednotky od A (A = 1, B = 2, C = 3, D = 4, atd.).

Viz také: Chdir, MkDir

Funkce MkDir

`Funkce MkDir(s: string) : Byte;`

Funkce se používá pro založení adresáře zadaného jména. Parametr s obsahuje cestu nového adresáře a jeho jméno. Poslední úsek cesty (to je jméno vytvářeného adresáře) nesmí být jméno existujícího souboru. Funkce vrací v případě úspěšnosti nulovou návratovou hodnotu.

Viz také: Chdir, GetDir

Funkce RmDir

```
function RmDir(s: string) : Byte;
```

Funkce vymazává adresář zadaný v parametru s. Adresář musí být prázdný. Pokud je adresář úspěšně vymazán, vrací funkce nulovou hodnotu.

Viz také: Chdir, GetDir, MkDir

Procedura ForEachFile

```
procedure ForEachFile(maska: string, ProcName: string,
                      rekurze: boolean);
```

Procedura umožňuje automatické vyhledání všech souborů, které vyhovují zadané masce DOSu v aktuálním adresáři (případně i ve všech podadresářích). Se všemi nalezenými soubory se potom provede zadaná operace.

Parametry:

maska - specifikace souborů pro vyhledávání dle konvencí DOSu. Například *.TXT vyhledá všechny soubory s příponou TXT.

Rekurze - zadává, zda se mají soubory vyhledávat i v podadresářích aktuálního adresáře. Pokud je zadáno False, provádí se hledání pouze v aktuálním adresáři.

ProcName - jméno procedury, která se provede pro každý nalezený soubor vyhovující zadané masce. Funkce musí mít následující deklaraci:

```
function name(fName: string, attr: Byte, time: Longint,
              size: Longint) : Boolean;
```

Funkce přebírá v parametru fName jméno souboru, atributy souboru v parametru attr, čas uložení souboru v parametru time a délku souboru v parametru size.

Funkce by měla vrátit hodnotu True, pokud se má pokračovat ve vyhledávání následujícího souboru. Pokud vrátí funkce hodnotu False, bude proces hledání procedurou ForEachFile předčasně ukončen.

Příklad:

```
VAR
   count : integer;
ENDVAR

PROCEDURE main
   forEachFile("*.IPS","printFileName",FALSE);
   WRITELN("počet:", count);
ENDPROC

PROCEDURE printFileName(fName: string, attr: byte, time: longint,
size: longint) : boolean;
   WRITELN(fName);
   RETURN TRUE;
ENDPROC
```

9.7. KNIHOVNA PRO PRÁCI S DATEM A ČASEM

Knihovna umožňuje především práci s datem a časem operačního systému. Možné je nejen čtení systémového data a času, ale také jeho nové nastavení.

GetDate - čte aktuální datum z operačního systému
SetDate - nastaví nové datum v operačním systému
GetTime - čte aktuální čas z operačního systému
SetTime - nastaví nový čas v operačním systému
JulToGreg - převádí juliánské datum na gregoriánské
GregToJul - převádí gregoriánské datum na juliánské

Procedura GetDate

```
procedure GetDate(var rok:word, var mesic:word, var den:word,
                  var dayOfWeek:word);
```

Procedura GetDate přebírá aktuální datum z operačního systému a nastavuje přebírané parametry. Parametr dayOfWeek bude obsahovat den v týdnu.

Viz také: SetDate, GetTime

Procedura SetDate

```
procedure SetDate(rok:word, mesic:word, den:word);
```

Procedura nastavuje v operačním systému aktuální datum na hodnotu dle předávaných parametrů.

Viz také: GetDate, SetTime

Procedura GetTime

```
procedure GetTime(var hodina:word,  var minuta:word,
                  var sekunda:word, var sec100:word);
```

Procedura přebírá z operačního systému do zadaných parametrů aktuální čas.

Viz také: SetTime, GetDate

Procedura SetTime

```
procedure setTime(hodina:word, minuta:word, sekunda:word,
                  sec100:word);
```

Procedura nastavuje v operačním systému aktuální čas podle zadaných parametrů.

Viz také: GetTime, SetDate

Funkce GregToJul

```
function GregToJul(měsíc:word, den:word, rok:word) : Longint;
```

Funkce vrací z poskytnutého gregoriánského data v parametrech juliánské datum.

Viz také: JulToGreg

Funkce JulToGreg

```
procedure JulToGreg(jul:Longint, var měsíc:word, var den:word,
                        var rok:word);
```

Procedura JulToGreg konvertuje Juliánské datum do tvaru gregoriánského data.

Viz také: GregToJul

9.8. INTERAKTIVNÍ KNIHOVNA

Interaktivní knihovna poskytuje sadu funkcí a procedur pro tvorbu dialogů s uživatelem. Velmi důležitá je možnost tvorby editačních formulářů. U formuláře je možné definovat rozměry, nadpis a umístění. Formulář má vždy v oblasti pravého horního rohu zobrazeny tlačítka OK a CANCEL. Dále je možné uživatelsky definovat další objekty formuláře:

- StaticText - statický text
- EditBox - editační textové pole
- CheckBox - tlačítko vypínače
- PushButton - příkazové tlačítko

Seznam procedur a funkcí pro definici formuláře

CreateForm	- definuje nový formulář
AddCheckBox	- definuje na formuláři vypínací box
AddEditBox	- definuje na formuláři editační pole
AddFrame	- definuje na formuláři rámeček
AddIcon	- definuje na formuláři zobrazení ikony (bitmapy)
AddPushButton	- definuje na formuláři příkazové tlačítko
AddSpeedButton	- definuje na formuláři tlačítko s bitmapou
AddStatic	- definuje na formuláři statický text
FormActionKey	- definuje proceduru pro zpracování stisknuté klávesy
FormDialog	- zobrazí formulář a umožní jeho editaci

Seznam procedur a funkcí pro editaci formuláře

DlgGetControlCheck	- čte stav vypínacího boxu
DlgGetControlText	- čte obsah editačního pole na formuláři
DlgSetControlCheck	- nastavuje hodnotu tlačítka ChexBox
DlgSetControlText	- nastavuje obsah textu nebo editačního pole
DlgDisableControl	- deaktivuje (znepřístupní) editační položku
DlgEnableControl	- aktivuje (zpřístupní) editační položku
DlgIsControlEnabled	- zjišťuje zda je editační položka přístupná
DlgHideControl	- ukryje (zneviditelní) editační pole
DlgShowControl	- zobrazí (zviditelní) editační pole
DlgIsControlVisible	- zjišťuje zda je editační položka viditelná
ChangeIcon	- změní zobrazovanou ikonu (bitmapu)
MoveIcon	- změní polohu ikony (bitmapy) na formuláři
MoveButton	- změní polohu tlačítka na formuláři

Seznam procedur a funkcí pro výběrové seznamy

Choose	- definice a editace výběrového seznamu
ClearChooseBox	- aktivuje nový výběrový seznam
AddChooseItem	- přidává novou volbu do výběrového seznamu
ChooseOption	- výběr volby z výběrového seznamu

Seznam ostatních procedur a funkcí

Answer	- vstup víceřádkového textu
MessageBox	- výpis hlášení na obrazovku

Text na plochu formuláře se zadává pomocí funkce AddStatic. Tento text nelze editovat, lze jej však

v případě potřeby změnit. Editační textové pole pro možnost vstupu dat se na formuláři definuje funkcí AddEditBox. Pro editaci logické hodnoty je možné funkcí AddCheckBox zadat na formuláři tlačítko vypínače. Příkazové tlačítko s možností vyvolání příslušné akce se na formulář zadá funkcí AddPushButton.

Jazyk INTER-PASCAL obsahuje procedury a funkce pro nastavení a čtení hodnot editačních polí (EditBox, CheckBox), jejich aktivaci a deaktivaci, ukrytí a zobrazení. To vám umožní rozsáhlé možnosti pro řízení a zpracování vstupu dat pomocí formuláře.

Další skupinou příkazů je možné provádět výběr ze seznamů. Používat je možné dva druhy výběrových seznamů. U prvního musíte znát již v době návrhu programu zobrazované volby, u druhého typu se seznam tvoří dynamicky za běhu programu. Interaktivní knihovna obsahuje i funkce pro vstup víceřádkového textu a zobrazení zpráv uživateli s možností volby několika různých tlačítek pro odpověď.

Upozornění: Pokud vytváříte formulář s definicí editačních položek addCheckBox, addStatic, addEditBox a funkcí addPushButton, které vracejí identifikátor ID, můžete použít uložení identifikátoru do proměnné, na kterou se budete později odkazovat bez nutnosti počítání řídící pozice editační položky na formuláři. To bude výhodné zejména v okamžiku, kdy budete měnit obsah formuláře, případně se dotazovat na stav tlačítek nebo obsah editačních polí.

Procedura CreateForm

```
procedure CreateForm(FormName: string, FormCaption: string,
                     x: integer, y: integer, šířka: integer,
                     výška: integer [,InitProc: string]);
```

Procedura CreateForm definuje nový vstupní formulář. Formulář je okno Windows, do kterého budete mít možnost definovat položky pro vstup údajů od uživatele.

Parametry:
FormName - jméno, identifikátor formuláře
FormCaption - titulek (záhlaví, nadpis) formuláře
x, y - levá horní pozice formuláře na obrazovce
šířka, výška - šířka a výška formuláře

InitProc - volitelný parametr který specifikuje jméno procedury, která bude provedena v okamžiku inicializace formuláře. V uvedené proceduře máte například možnost provedení akcí související s inicializací používaných proměnných.

Viz také:
AddStatic, AddEditBox, AddCheckBox, AddPushButton, FormDialog

Funkce AddCheckBox

```
function  AddCheckBox(FormName: string, VarIn: string,
                      VarOut: string, Záhlaví:  string,
                      X: integer, Y: integer, šířka: integer,
                       výška: integer [, ProcName: string]) : word;
```

Funkce AddCheckBox se používá pro přidání přepínacího boxu do formuláře. Lze zadat proměnnou VarIn, která obsahuje stav boxu v okamžiku aktivace, a proměnnou VarOut, do které se uloží stav tlačítka v době ukončení editace formuláře. Pokud je provedena změna stavu boxu, provede se procedura ProcName, ve které se mohou provést například kontroly vstupních dat.

Parametry:
FormName - jméno, identifikátor formuláře
VarIn - proměnná, jejíž hodnota nastavuje v okamžiku vytvoření formuláře stav CheckBoxu. Pokud použijete "", znamená to, že vstupní proměnná není použita.
VarOut - proměnná, do které bude převeden stav CheckBoxu v okamžiku ukončení editace formuláře tlačítkem "OK".
Záhlaví - text který se zobrazí vpravo od CheckBoxu
X, Y - souřadnice pozice, na které se zobrazí ChexBox
šířka, výška - velikost CheckBoxu
ProcName - volitelné jméno procedury která bude provedena vždy, když uživatel stiskne ChexBox

Návrat:
Funkce vrací řídící identifikátor ID, který může být použit v dialogu celkové editace formuláře za použití procedur DlgEnableControl, DlgShowControl a dalších.

Viz také:
CreateForm, AddStatic, AddEditBox, AddPushButton, FormDialog

Funkce AddEditBox

```
function AddEditbox(FormName: string, VarIn: string,
                    VarOut: string, X: integer, Y: integer,
                    šířka: integer, výška: integer
                    [, ProcName: string]) : word;
```

Procedura AddEditBox přidá na formulář nové editační pole. Lze zadat proměnnou VarIn, která obsahuje stav editačního pole v okamžiku aktivace, a proměnnou VarOut, do které se uloží stav editačního pole v době ukončení editace formuláře. Pokud je provedena změna obsahu editačního pole, provede se procedura ProcName, ve které se mohou provést například kontroly vstupních dat.

Parametry:
FormName - jméno, identifikátor formuláře
VarIn - vstupní proměnná, jejíž hodnota se předává do editace, pokud se zadá "", znamená to, že vstupní proměnná není použita
VarOut - výstupní proměnná, do které se převede editovaný údaj v okamžiku ukončení editace formuláře tlačítkem "OK"
X, Y - souřadnice levého horního rohu editačního pole na formuláři
šířka, výška - velikost editované položky na formuláři
ProcName - jméno volitelné procedury, která bude volaná při změně editačního pole. Procedura může obsahovat například kontrolu rozsahu platnosti zadaných dat.

Návrat:
Funkce vrací řídící identifikátor ID, který může být použit v dialogu celkové editace formuláře za použití procedur DlgEnableControl, DlgShowControl a dalších.

Viz také:
CreateForm, AddStatic, AddCheckBox, AddPushButton, FormDialog

Procedura AddFrame

```
procedure AddFrame(formName: string, X: integer, Y: integer,
                   šířka: integer, výška: integer)
```

Procedura přidá na formulář rámeček, který umožní vhodně rozdělit plochu formuláře na jednotlivé části. Pozor na to, že ačkoliv nemá definovaný rámeček možnost dodatečné změny svých parametrů, zvyšuje také hodnotu řídícího identifikátoru ID.

Parametry:

FormName	- jméno, identifikátor formuláře
X, Y	- souřadnice levého horního rohu rámečku na formuláři
šířka, výška	- velikost zobrazovaného rámečku

Funkce AddIcon

```
function AddIcon(formName: string, ProcName: string
                        bitmapa : string, X: integer,
                  Y: integer, šířka: integer, výška: integer
                  [, getX: string, getY: string] ) : word;
```

Funkce AddIcon se používá na přidání ikony (načtené ze souboru *.ICO), případně bitmapového obrázku (soubor typu *.BMP) na plochu formuláře.

Parametry:
FormName - jméno, identifikátor formuláře
ProcName - jména procedury, která bude aktivována, pokud klikne uživatel myší na ploše ikony (bitmapy)
bitmapa - jméno souboru typu BMP nebo ICO
X, Y - souřadnice levého horního rohu tlačítka na formuláři
šířka, výška - velikost zobrazované bitmapy (ikony)
getX,getY - jména proměnných, do kterých budou převedeny souřadnice pozice na bitmapě po kliknutí myší na její plochu

Návratová hodnota:
Funkce vrací řídící identifikátor ID, který může být použit v dialogu celkové editace formuláře. Například pro změnu bitmapy nebo její přesun na novou pozici.

Zadaná procedura umožňuje po kliknutí myší na plochu bitmapy provést libovolnou další akci. Zadáte-li v deklaraci funkce AddIcon jména proměnných (parametry getX a getY), budou do nich nejprve převedeny souřadnice bodu, na který bylo muší kliknuto. Proměnné musí být deklarovány jako typ integer.
Zobrazenou bitmapu je možné kdykoliv změnit procedurou ChangeIcon. To vám umožní například programovat jednoduché hry, kdy se plocha formuláře mění podle prováděných akcí.

Funkce AddPushButton

```
function AddPushButton(FormName: string, ProcName: string,
                       záhlaví: string, X: integer,
                       Y: integer,  šířka: integer,
                       výška: integer) : word;
```

Funkce AddPushButton se používá pro přidání nového povelového tlačítka do formuláře. Pokud v průběhu editace uživatel tlačítko stiskne, je provedena procedura uvedená v parametru procName.

Parametry:
FormName - jméno, identifikátor formuláře
ProcName - jméno procedury která bude aktivovaná, pokud stiskne uživatel tlačítko
Záhlaví - text který se zobrazí na tlačítku
X, Y - souřadnice levého horního rohu tlačítka na formuláři
šířka, výška - velikost povelového tlačítka

Návratová hodnota:
Funkce vrací řídící identifikátor ID, který může být použit v dialogu celkové editace formuláře za použití procedur DlgEnableControl, DlgShowControl a dalších.

Viz také:
CreateForm, AddStatic, AddEditBox, AddCheckBox, FormDialog

Funkce AddSpeedButton

```
function AddSpeedButton(formName: string, ProcName: string
                        bitmapa : string, X: integer,
                        Y: integer, šířka: integer,
                        výška: integer) : word;
```

Funkce AddSpeedButton se používá na přidání ovládacího tlačítka s bitmapou (místo textu) načtenou ze souboru typu BMP. Pokud tlačítko na formuláři stisknete, bude provedena procedura uvedená v parametru ProcName.

Parametry:

FormName - jméno, identifikátor formuláře
ProcName - jména procedury, která bude aktivována, pokud stiskne uživatel tlačítko
bitmapa - jméno souboru typu BMP, který se zobrazí v tlačítku
X, Y - souřadnice levého horního rohu tlačítka na formuláři
šířka, výška - velikost povelového tlačítka

Návratová hodnota:

Funkce vrací řídící identifikátor ID, který může být použit v dialogu celkové editace formuláře. Například pro ukrytí tlačítka, nebo jeho přesun na novou pozici.

Bitmapu pro tlačítko můžete vytvořit v libovolném grafickém editoru, který je schopen ukládat výsledky do souborů typu BMP. Doporučená velikost bitmapy pro použití jako povelové tlačítko je 24x24 až 32x32 bodů. Se systémem KLONDAIK se dodává několik souborů s vyobrazením základních tlačítek.

Pokud nebude soubor s bitmapou nalezen, nebude hlášena systémem žádná chyba, tlačítko zůstane v takovém případě prázdné. Bitmapa je v tlačítku umístěna v jeho středu a může být proto menší, než je tlačítko definováno.

Funkce AddStatic

```
function AddStatic(FormName: string, Záhlaví: string,
                   X: integer, Y: integer, šířka: integer,
                   výška: integer) : word;
```

Funkce AddStatic zobrazí na formuláři statický, neměnný text na zadané pozici.

Parametry:
FormName - jméno, identifikátor formuláře
Záhlaví - zobrazovaný text
X, Y - levá horní souřadnice textu na formuláři
šířka, výška - velikost textu

Návrat:
Funkce vrací řídící identifikátor ID, který může být použit v dialogu celkové editace formuláře za použití procedur DlgEnableControl, DlgShowControl a dalších.

Viz také:
CreateForm, AddEditBox, AddCheckBox, AddPushButton, FormDialog

Procedura FormActionKey

```
procedure FormActionKey(FormName: string, ProcName: string
                        [, Key: string])
```

Procedura přidá formuláři možnost reagovat na stisk libovolné klávesy vykonáním zadané procedury. Pozor na to, že ačkoliv nemá definovaný rámeček možnost dodatečné změny svých parametrů, zvyšuje také hodnotu řídícího identifikátoru ID.

Parametry:
FormName - jméno, identifikátor formuláře
ProcName - jména procedury, která bude aktivována, po stisku libovolné klávesy
Key - jméno proměnné, do které se předá kód stisknuté klávesy

Proceduru je vhodné používat například při programování her, kdy umožňuje reagovat na stisk libovolné klávesy a podle kódu stisknuté klávesy provést příslušnou akci. Není vhodné používat proceduru, pokud máte na formuláři definovány editační pole. Vzhledem k tomu, že pokud máte na formuláři definovány tlačítka typu PuschButton, způsobí stisk na klávesy s šipkami přechod na další tlačítko, není vhodné používat současně na formuláři ani tlačítka typu PuschButton.

Při použití procedury FormActionKey nejsou na formuláři zobrazována tlačítka pro ukončení editace formuláře "Ok" a "Konec". Ukončení editace je proto možné provést pouze systémovou uzavírací ikonou v horní liště okna. Tím je umožněno, aby bylo možné celou plochu formuláře použít pro vlastní definici obsahu formuláře. Zvláště při programování her by uzavírací tlačítka formuláře působila zvlášť rušivě. Pokud budete chtít této vlastnosti (nezobrazování uzavíracích tlačítek) použit i pro použití v běžném formuláři, zadejte v definici procedury FormActionKey místo jména procedury prázdný řetězec. Pozor na to, že vzhledem k tomu, že není možné stisknout tlačítko "Ok", vrací funkce FormDialog při použití procedury FormActionKey vždy hodnotu false.

Procedura FormActionKey předá vámi definované proceduře hodnotu stisknuté klávesy. Hodnota se předá do zadané proměnné, která musí být deklarována s typem word. Kód stisknuté klávesy je stejný jako u systémové funkce GetKey.

Výběr kódů kláves:

klávesa	kód	klávesa	kód
šipka nahoru	38	F1	112
šipka dolů	40	F2	113
šipka vpravo	39	F3	114
šipka vlevo	37	F4	115
Home	36	F5	116
End	35	F6	117
PageUp	33	F7	118
PageDown	34	F8	119
Insert	45	F9	120
Delete	46	F10	121

Funkce FormDialog

```
function FormDialog(formName : string) : boolean;
```

Funkce zobrazuje nadefinovaný formulář ve tvaru dialogového okna a je umožněna uživatelská editace editačních položek. Formulář musí být ještě před zadáním funkce FormDialog definován procedurou CreateForm a musí být zadány objekty formuláře příkazy AddStatic (statický text), AddEditBox (editační pole), AddCheckBox (vypínač) a AddPushButton (příkazové tlačítko). Po ukončení editace formuláře vrací funkce logickou hodnotu udávající způsob ukončení editace.

Parametry:
FormName - jméno formuláře, které se má zobrazit jako dialog

Návrat:
Pokud byla editace formuláře ukončena stiskem tlačítka "OK", vrací funkce True, jinak vrací False.

Viz také:
CreateForm, AddStatic, AddEditBox, AddCheckBox, AddPushButton

Funkce DlgGetControlCheck

```
function DlgGetControlCheck(ctlID: word) : boolean;
```

Funkce obnovuje (aktualizuje) stav proměnné představující na formuláři hodnotu CheckBoxu.

Parametry:
CtlID - řídící identifikátor ID stanovující pořadí, ve kterém jste přidávali objekty do formuláře po jeho aktivaci procedurou CreateForm.

Příklad použití:

```
CreateForm("MyForm",...);
AddStatic("MyForm",...);      {hodnota ID bude nastavena na 1}
AddCheckBox("MyForm",...);    {hodnota ID bude nastavena na 2}
AddPushButton("MyForm",...); {hodnota ID bude nastavena na 3}
FormDialog("MyForm")
```

Pokud by jste chtěli převést stav CheckBoxu z formuláře do proměnné MyVar, zadejte:

```
MyVar := DlgGetControlCheck(2)
```

Viz také:
DlgSetControlCheck

Funkce DlgGetControlText

```
function DlgGetControlText(ctrlId: word) : string;
```

Funkce obnovuje (aktualizuje) stav proměnné představující na formuláři hodnotu editačního pole EditBox.

Parametry:
CtlID - řídící identifikátor ID stanovující pořadí, ve kterém jste přidávali objekty do formuláře po jeho aktivaci procedurou CreateForm.

Příklad použití:

```
CreateForm("MyForm",...);
AddStatic("MyForm",...);      {hodnota ID bude nastavena na 1}
AddEditBox("MyForm",...);     {hodnota ID bude nastavena na 2}
AddPushButton("MyForm",...); {hodnota ID bude nastavena na 3}
FormDialog("MyForm")
```

Pokud by jste chtěli převést text prvního editačního pole na formuláři do proměnné MyVar, zadejte:

```
MyVar := dlgGetControlText(2)
```

Viz také:
DlgSetControlText

Procedura DlgSetControlCheck

```
procedure DlgSetControlCheck(ctrlId : word; TheData :boolean);
```

Procedura nastavuje hodnotu tlačítka CheckBox zadaného identifikátorem ID.

Parametry:
CtlID - řídící identifikátor ID stanovující pořadí, ve kterém jste přidávali objekty do formuláře po jeho aktivaci procedurou CreateForm.

TheData - stav (True nebo False), který chcete nastavit

Příklad použití:

```
CreateForm("MyForm",...);
AddStatic("MyForm",...);      {hodnota ID bude nastavena na 1}
AddCheckBox("MyForm",...);    {hodnota ID bude nastavena na 2}
AddPushButton("MyForm",...); {hodnota ID bude nastavena na 3}
FormDialog("MyForm")
```

Pro nastavení výše uvedeného tlačítka CheckBox můžete použít následující volání procedury:

```
DlgSetControlCheck(2, TRUE)
```

Viz také:
DlgGetControlCheck

Procedura DlgSetControlText

```
procedure DlgSetControlText(ctrlId : word; Text : string);
```

Procedura nastavuje text objektu na formuláři zadaného identifikátorem ID.

Parametry:
CtlID - řídící identifikátor ID stanovující pořadí, ve kterém jste přidávali objekty do formuláře po jeho aktivaci procedurou CreateForm.
Text - nový text, který bude zobrazen

Příklad použití:

```
CreateForm("MyForm",...);
AddStatic("MyForm",...);      {hodnota ID bude nastavena na 1}
AddEditBox("MyForm",...);     {hodnota ID bude nastavena na 2}
AddPushButton("MyForm",...); {hodnota ID bude nastavena na 3}
FormDialog("MyForm")
```

Pokud budete chtít změnit výše uvedený statický text, máte možnost tak učinit následujícím voláním procedury:

```
DlgSetControlText(1,"Nový Text")
```

Viz také:
DlgGetControlText

Procedura DlgDisableControl

```
procedure DlgDisableControl(ctrlId : word);
```

Procedura deaktivuje editační položku, jejíž identifikátor je specifikován v argumentu procedury. Po deaktivaci nebude možné položku editovat. Obnova možnosti editace položky se provede procedurou DlgEnableControl.

Parametry:
CtlID - řídící identifikátor ID stanovující pořadí, ve kterém jste přidávali objekty do formuláře po jeho aktivaci procedurou CreateForm.

Příklad použití:

```
CreateForm("MyForm",...);
AddStatic("MyForm",...);      {hodnota ID bude nastavena na 1}
AddEditBox("MyForm",...);     {hodnota ID bude nastavena na 2}
AddPushButton("MyForm",...); {hodnota ID bude nastavena na 3}
FormDialog("MyForm")
```

Pokud by jste chtěli deaktivovat první editační pole na formuláři, zadejte následující kód:

```
DlgDisableControl(2);
```

Viz také:
DlgIsControlEnabled, DlgEnableControl

Procedura DlgEnableControl

```
procedure DlgEnableControl(ctlID : word);
```

Procedura aktivuje editační položku, jejíž identifikátor je specifikován v argumentu procedury.

Parametry:
CtlID - řídící identifikátor ID stanovující pořadí, ve kterém jste přidávali objekty do formuláře po jeho aktivaci procedurou CreateForm.

Příklad použití:

```
CreateForm("MyForm",...);
AddStatic("MyForm",...);      {hodnota ID bude nastavena na 1}
AddEditBox("MyForm",...);     {hodnota ID bude nastavena na 2}
AddPushButton("MyForm",...); {hodnota ID bude nastavena na 3}
FormDialog("MyForm")
```

Pokud by jste chtěli aktivovat první editační pole na formuláři, zadejte:

```
DlgEnableControl(2);
```

Viz také:
DlgIsControlEnabled, DlgDisableControl

Funkce DlgIsControlEnabled

```
function DlgIsControlEnabled(ctrlId:word) : boolean;
```

Funkce určuje, zda je editační položka formuláře přístupná, nebo blokovaná pro editaci. Tuto vlastnost lze změnit procedurami DlgDisableControl a DlgEnableControl.

Parametry:
CtlID - řídící identifikátor ID stanovující pořadí, ve kterém jste přidávali objekty do formuláře po jeho aktivaci procedurou CreateForm.

Příklad použití:

```
CreateForm("MyForm",...);
AddStatic("MyForm",...);      {hodnota ID bude nastavena na 1}
AddEditBox("MyForm",...);     {hodnota ID bude nastavena na 2}
AddPushButton("MyForm",...); {hodnota ID bude nastavena na 3}
FormDialog("MyForm")
```

Pokud by jste chtěli zjistit, zda je první editační okno na formuláři aktivované (přístupné) nebo deaktivované (nepřístupné), můžete se ptát následujícím dotazem:

```
IF (DlgIsControlEnabled(2)) then
   ...
ENDIF
```

Viz také:
DlgDisableControl, DlgEnableControl

Procedura DlgHideControl

```
procedure DlgHideControl(ctlID : word);
```

Procedura ukryje (zneviditelní) editační pole na formuláři. Nové zviditelnění položky se potom provede procedurou DlgShowControl.

Parametry:
CtlID - řídící identifikátor ID stanovující pořadí, ve kterém jste přidávali objekty do formuláře po jeho aktivaci procedurou CreateForm.

Příklad použití:

```
CreateForm("MyForm",...);
AddStatic("MyForm",...);      {hodnota ID bude nastavena na 1}
AddEditBox("MyForm",...);     {hodnota ID bude nastavena na 2}
AddPushButton("MyForm",...); {hodnota ID bude nastavena na 3}
FormDialog("MyForm")
```

Pokud by jste chtěli skrýt první editační pole na formuláři, zadejte následující kód:

```
DlgHideControl(2)
```

Viz také:
DlgIsControlVisible, DlgShowControl

Procedura DlgShowControl

```
procedure DlgShowControl(ctlID : word);
```

Procedura nastavuje aktivní (editovaný) objekt na formuláři zadaný identifikátorem ID. Tak je možné změnit pořadí editace položek, případně vynutit si zadání požadované položky po kontrole správnosti zadaných dat.

Parametry:
CtlID - řídící identifikátor ID stanovující pořadí, ve kterém jste přidávali objekty do formuláře po jeho aktivaci procedurou CreateForm.

Příklad použití:

```
CreateForm("MyForm",...);
AddStatic("MyForm",...);      {hodnota ID bude nastavena na 1}
AddEditBox("MyForm",...);     {hodnota ID bude nastavena na 2}
AddPushButton("MyForm",...); {hodnota ID bude nastavena na 3}
FormDialog("MyForm")
```

Pokud chcete nastavit aktivní (aktuálně editovaný) objekt výše uvedený EditBox, můžete tak učinit následujícím voláním procedury:

```
DlgShowControl(2)
```

Viz také:
DlgIsControlVisible, DlgHideControl

Funkce DlgIsControlVisible

```
function DlgIsControlVisible(ctrlId : word) : boolean;
```

Funkce určuje, zda je viditelná editační položka zadaná identifikátorem ID.

Parametry:
CtlID - řídící identifikátor ID stanovující pořadí, ve kterém jste přidávali objekty do formuláře po jeho aktivaci procedurou CreateForm.

Příklad použití:

```
CreateForm("MyForm",...);
AddStatic("MyForm",...);      {hodnota ID bude nastavena na 1 }
AddEditBox("MyForm",...);     {hodnota ID bude nastavena na 2 }
AddPushButton("MyForm",...); {hodnota ID bude nastavena na 3 }
FormDialog("MyForm")
```

Pokud potřebujete zjistit, zda je editační položka viditelná, můžete použít následující dotaz:

```
IF (DlgIsControlVisible(2)) then
   ...
ENDIF
```

Viz také:
DlgShowControl, DlgHideControl

Procedura ChangeIcon

```
procedure ChangeIcon(CtrlID: word, file: string);
```

Procedura umožňuje změnit zobrazenou ikonu (bitmapu) umístěnou na formuláři. Použití procedury je velmi vhodné například při programování her apod.

Parametry:
CtrlID - řídící identifikátor udávající pořadí, ve kterém jste přidávali objekty na formulář po jeho aktivaci procedurou CreateForm.
file - jméno souboru s ikonou, případně bitmapou

Zobrazit je možné pouze soubory typu BMP (bitmapy), případně ICO (ikony). Pokud není soubor nalezen, není hlášena žádná chyba.

Příklad použití:

```
CreateForm("MyForm", ...);
AddStatic ("MyForm", ...);     {hodnota ID bude nastavena na 1}
AddIcon("MyForm", ...);        {hodnota ID bude nastavena na 2}
AddPuschButton("MyForm",...); {hodnota ID bude nastavena na 3}
FormDialog("MyForm");
```

Pokud budete chtít změnit zobrazovanou ikonu, zadejte například:

```
ChangeIcon(2, "NOVA.ICO");
```

Procedura MoveIcon

```
procedure MoveIcon(CtrlId: word, x: integer, y: integer);
```

Procedura umožňuje změnit polohu ikony (bitmapy) umístěné na formuláři. Použití procedury je velmi vhodné například při programování her apod.

Parametry:
CtrlID - řídící identifikátor udávající pořadí, ve kterém jste přidávali objekty na formulář po jeho aktivaci procedurou CreateForm.
x, y - nové souřadnice polohy ikony, případně bitmapy (její velikost nelze změnit)

Příklad použití:

```
CreateForm("MyForm", ...);
AddStatic ("MyForm", ...);     {hodnota ID bude nastavena na 1}
AddIcon("MyForm", ...);        {hodnota ID bude nastavena na 2}
AddPuschButton("MyForm",...); {hodnota ID bude nastavena na 3}
FormDialog("MyForm");
```

Pokud budete chtít změnit polohu ikony na nové souřadnice, zadejte:

```
MoveIcon(2, 100, 150);
```

Procedura MoveButton

```
procedure MoveButton(CtrlId: word, x: integer, y: integer);
```

Procedura umožňuje změnit polohu tlačítka typu PuschButton, nebo SpeedButton na formuláři. Použití je možné například při programování her apod.

Parametry:
CtrlID - řídící identifikátor udávající pořadí, ve kterém jste přidávali objekty na formulář po jeho aktivaci procedurou CreateForm.
x, y - nové souřadnice polohy tlačítka (jeho velikost nelze změnit)

Příklad použití:

```
CreateForm("MyForm", ...);
AddStatic ("MyForm", ...);   {hodnota ID bude nastavena na 1}
AddIcon("MyForm", ...);      {hodnota ID bude nastavena na 2}
AddPuschButton("MyForm",...);{hodnota ID bude nastavena na 3}
FormDialog("MyForm");
```

Pokud budete chtít změnit polohu tlačítka na novou polohu, zadejte:

```
MoveButton(3, 100, 150);
```

Funkce Choose

```
function Choose(message: string, default: string, volba1:
string, volba2: string,.. volbaN: string);
```

Funkce zobrazí dialogové okno se seznam voleb a vrací hodnotu, udávající kterou volbu uživatel zvolil.

Parametr message udává zprávu která bude zobrazena nad seznamem.

Parametr default určuje, která volba bude na počátku výběru zvýrazněna (nemusí být první).

Parametry volba1 až volbax jsou volby, které se zobrazí v okně seznamu v uvedeném pořadí.

Všimněte si, že standardní (default) volba musí být zadána i v seznamu, protože jinak by jazyk INTER-PASCAL nebyl schopen poznat, která volba má být nastavena v okamžiku inicializace.

Funkce vrací hodnotu -1 pokud stiskl uživatel tlačítko pro zrušení. Pokud je vrácena jiná hodnota, provedl uživatel výběr jedné z nabízených možností a hodnota uvádí pořadí vybrané možnosti. Hodnota 0 přitom uvádí první volbu.

Příklad:

```
IF (Choose("vyberte si:", "Jablko", "banán", "hruška") <> -1
   MessageBox("Vybral jste si" + IT, "Výsledek", mb_ok)
ELSE
   MessageBox("Stiskl jste Cancel", "Výsledek", mb_ok);
ENDIF
```

V tomto příkladu je zobrazen v okně seznam s třemi volbami (pomeranč, jablko a banán). Dialogové okno zobrazuje současně zprávu o možnosti výběru ("vyberte si:"). Standardní, předvolenou možností je v tomto případě volba "jablko". Pokud uživatel zruší operaci výběru, bude vrácena hodnota -1. Jinak bude identifikátor IT obsahovat volbu uživatele.

Všimněte si, že pokud neznáte předem počet položek pro výběr, budete muset místo funkce Choose použít kompletu procedur ClearChooseBox (inicializace), AddChooseItem (zadání položek) a ChooseOption (výběr položky).

Viz také:
ClearChooseBox, AddChooseItem, ChooseOption

Procedura ClearChooseBox

```
procedure ClearChooseBox;
```

Procedura ClearChooseBox vytváří (aktivuje) novou tabulku pro výběr, ze které budete mít následně možnost vybrat si z nabízených položek. Provádí inicializaci tabulky a nastavuje počet položek na nulu. Naplnění tabulky se provádí procedurou AddChooseBox, vlastní výběr je prováděn procedurou ChooseOption. Pokud znáte přesný počet položek seznamu a nepotřebujete během programu přidávat další, použijte raději proceduru Choose.

```
PROCEDURE ChooseFromDB;
VAR
   row: string;
ENDVAR
   ClearChooseBox                      {musí být použito}
   WHILE (not databaseEof) do begin
      row := DoDatabaseRead;
      addChooseItem(row);              {přidání položky}
   ENDWHILE;
   IF (chooseOption(                   {výběr položky}
      "Vyberte volbu, o které chcete více informací",
                  "Macintosh") <> -1) then
      messageBox("Vybral jste si:" + IT "Výsledek",mb_ok)
   ELSE
      messageBox("volba nebyla vybrána", "Výsledek",mb_ok)
   ENDIF
ENDPROC
```

Viz také:
AddChooseItem, ChooseOption, Choose

Procedura AddChooseItem

```
procedure AddChooseItem(item: string);
```

Procedura AddChooseItem přidává do dialogu výběrové tabulky novou volbu a zvyšuje tím počet nabízených voleb. Tuto proceduru je vhodné použít v případě, že potřebujete, aby si uživatel zvolil jednu z nabízených možností uvedených v seznamu ačkoliv při návrhu programu neznáte, kolik možností ve skutečnosti bude. Pokud znáte přesný počet voleb již při zápisu programu, použijte raději funkci Choose.

Procedura je využitelná pouze v kompletu s procedurami ClearChooseBox (provede počáteční nastavení) a ChooseOption (výběr jedné z možností). Používá se například, pokud potřebujete převzít volbu ze souboru.

Příklad:

```
PROCEDURE ChooseFromDB;
VAR
   row: string;
ENDVAR
   ClearChooseBox                    {musí být použito !}
   WHILE (not databaseEof) do begin
      row := DoDatabaseRead;
      addChooseItem(row);            {přidání položky}
   ENDWHILE;
   IF (chooseOption(                {výběr položky}
      "Vyberte volbu, o které chcete více informací",
                  "OZOGAN") <> -1) then
      messageBox("Vybral jste si:" + IT, "Výsledek", mb_ok)
   ELSE
      messageBox("volba nebyla vybrána", "Výsledek", mb_ok)
   ENDIF
ENDPROC
```

Viz také:
ClearChooseBox, ChooseOption, Choose

Funkce ChooseOption

```
function ChooseOption(message: string, default: string);
```

Používá se pro výběr volby ze seznamu, o kterém předem nevíte, kolik bude obsahovat položek. Funkce se používá společně s funkcemi ClearChooseBox (inicializace) a AddChooseItem (přidání položky). Pokud znáte předem počet hodnot pro výběr, je výhodnější použít funkci Choose.

Funkce ChooseOption zobrazí dialogové okno s Vaší zprávou message a umožní Vám listovat všemi volbami, které byly zadány funkcí AddChooseItem. Aktuálně vybraná položka je zvýrazněna.

Pokud zruší uživatel akci výběru stiskem tlačítka "CANCEL", vrací funkce hodnotu -1, jinak vrací index na vybranou položku. První položka má přitom index 0.

```
PROCEDURE ChooseFromDB;
VAR
   row : string;
ENDVAR
   ClearChooseBox                        {musí být použito}
   WHILE (not databaseEof) do begin
      row := DoDatabaseRead;
      addChooseItem(row);                {přidání položky}
   ENDWHILE;
   IF (ChooseOption(                     {výběr položky}
      "Vyberte volbu, o které chcete více informací",
                  "Macintosh") <> -1) then
      messageBox("Vybral jste si:" + IT, "Výsledek",mb_ok)
   ELSE
      messageBox("volba nebyla vybrána", "Výsledek",mb_ok)
   ENDIF
ENDPROC
```

Viz také:
ClearChooseBox, AddChooseItem, Choose

Funkce Answer

```
function Answer(popis: string) : boolean;
```

Funkce zobrazí uživateli v samostatném okně text předaný v parametru popis spolu s editačním víceřádkovým textovým oknem a tlačítky OK a CANCEL. Uživatel má možnost zapsat libovolný víceřádkový text. Pokud uživatel ukončí dialog stiskem tlačítka "OK", je funkcí předána hodnota True, jinak False. V systémové proměnné IT je přitom navíc předávaná uživatelova odpověď k dalšímu použití.

Příklad:

```
IF (answer("zadejte vaši adresu")) then
   WRITELN(IT);
ENDIF
```

Funkce MessageBox

```
function MessageBox(Text: string;  Titul: string;
                    Příznaky: word) : integer;
```

Funkce MessageBox se používá pro výpis hlášení na obrazovku s možností odpovědi od uživatele. Jako parametr funkce je možné zadat typ ikony, která bude zobrazena společně se zprávou. Současně je možné zadat tlačítka, která se zobrazí pro odpověď uživatele.

Parametry:
Text - textová zpráva která bude zobrazena
Titul - titulek okna se zprávou
Příznaky - kombinace příznaků udávající zobrazená tlačítka a ikonu. Příznak zadejte buď součtem hodnot, nebo výpisem konstant:

hodnota	konstanta	význam
0	MB_OK	zobrazí pouze tlačítko OK
1	MB_OKCANCEL	zobrazí tlačítka OK a CANCEL
2	MB_ABORTRETRYIGNORE	zpráva obsahuje tlačítka ABORT, RETRY a IGNORE
3	MB_YESNOCANCEL	zobrazí tlačítka ANO, NE a CANCEL
4	MB_YESNO	zobrazí tlačítka ANO a NE
5	MB_RETRYCANCEL	zobrazí tlačítka RETRY a CANCEL
16	MB_ICONSTOP	zobrazí v okně ikonu STOP
32	MB_ICONQUESTION	zobrazí v okně ikonu s otazníkem
48	MB_ICONEXCLAMATION	zobrazí v okně ikonu vykřičníku
64	MB_ICONINFORMATION	zobrazí v okně ikonu I v kruhu

Návrat:
Funkce vrací hodnotu nula, pokud není dost paměti pro vytvoření okna se zprávou. Jinak vrací funkce jednu z následujících hodnot:

hodnota	konstanta	význam
1	IDOK	bylo vybráno tlačítko OK
2	IDCANCEL	bylo vybráno tlačítko CANCEL
3	IDABORT	bylo vybráno tlačítko ABORT
4	IDRETRY	bylo vybráno tlačítko RETRY (opakovat)
5	IDIGNORE	bylo vybráno tlačítko IGNORE
6	IDYES	bylo vybráno tlačítko ANO
7	IDNO	bylo vybráno tlačítko NE

Pokud bude okno se zprávou obsahovat tlačítko ABORT, bude vrácena hodnota IDCANCEL i v případě stisku tlačítka ESC. Pokud neobsahuje okno se zprávou tlačítko ABORT, nebude mít stisk klávesy ESC žádný význam.

Příklad:

Pokud chcete zobrazit zprávu "Provést výpočet?" v okně s titulem "Dotaz", se zobrazením ikony otazníku a s tlačítky ANO a NE, můžete zadat příkaz:

```
MessageBox('výpočet?','Dotaz',MB_YESNO+MB_ICONQUESTION);
```

nebo

```
MessageBox('výpočet?','Dotaz',36);  {4 + 32 = 36}
```

9.9. GRAFICKÁ KNIHOVNA

V grafické knihovně jsou definovány procedury a funkce sloužící k výstupu do grafického okna. Do okna je možné kreslit čáry, čtverce, kružnice a elipsy, čtverce, obdélníky a trojúhelníky. Možné je nastavit barvu, druh a styl čáry. Stejně tak je možné nastavit barvu ploch. Plochu grafického okna je možné ukládat a načítat v souborech typu *.BMP.

Souřadnicový systém
Definice barev
Definice čar
Definice výplně ploch
Definice fontu

Příkazy a funkce grafické knihovny:

Arc	- kreslí křivku, část elipsy
Ellipse	- kreslí elipsu nebo kružnici
GetMaxX	- vrací velikost grafického okna na ose x (šířka)
GetMaxY	- vrací velikost grafického okna na ose y (výška)
ImageBrushColor	- nastaví barvu výplně ploch
ImageBrushStyle	- nastaví styl vyplňování ploch
ImageClear	- vymaže obsah grafického okna
ImageFontColor	- nastaví barvu fontu
ImageFontName	- nastaví typ fontu
ImageFontSize	- nastaví velikost fontu
ImageFontStyle	- nastaví styl fontu (bold, italic, podtržení..)
ImageFromClip	- převede obsah schránky na zadanou pozici
ImageInit	- inicializuje grafické okno
ImageLoad	- načte soubor do grafického okna
ImageMove	- zkopíruje část grafického okna na novou pozici
ImagePenColor	- nastaví barvu kreslených čar
ImagePenStyle	- nastaví druh čáry (plná, tečkovaná, ..)
ImagePenWidth	- nastaví sílu čáry
ImagePrint	- výstup grafického okna na tiskárnu
ImageSave	- zapíše grafické okno do souboru
ImageSetFont	- vyvolá dialog pro nastavení fontu
ImageToClip	- uloží výřez grafického okna do schránky
Line	- kreslí úsečku dle zadaných souřadnic (počátek, konec)
LineTo	- kreslí úsečku z aktuální pozice do zadaného bodu
MoveTo	- přesune pozici grafického ukazatele
Pie	- kreslí kruhovou výseč
Point	- vykreslí bod na zadané souřadnici
Rectangle	- kreslí pravoúhelník nebo čtverec
RoundRect	- kreslí pravoúhelník se zaoblenými hranami
TextHeight	- vrací výšku textu v bodech
TextOut	- vypíše zadaný text do grafického okna
TextWidth	- vrací šířku textu v bodech
Triangle	- nakreslí trojúhelník

Souřadnicový systém

V grafickém okně se zápis a kreslení provádí v souřadnicovém systému. Souřadnice se zadávají v bodech. Souřadnicový systém je vztažen k levému hornímu rohu, který má souřadnici 0,0. Hodnoty ve směru osy X narůstají směrem doprava, hodnoty ve směru osy Y narůstají směrem dolů. Při zápisu souřadnice se uvádí nejprve osa x, potom osa y. Je přitom možné zadávat příkazy pro kreslení mimo plochu grafického okna, zobrazí se však pouze ta část, která je obsažena maximálními souřadnicemi grafického okna.

V grafickém okně je definován tzv. grafický ukazatel, který zaznamenává pozici vykreslení posledního bodu. Poloha grafického ukazatele se při použití některých procedur automaticky mění. Je možné ji nastavit i z programu procedurou MoveTo.

Činnost grafického okna je možné si představit jako malířské plátno definovaných rozměrů, na které se kreslí perem (anglicky pen). Větší plochy je možné vybarvit štětcem (anglicky brush). Kreslí se přitom vždy nastavenou barvou. Aby bylo možné zadávat jednoduše příkazy pro zápis do grafického okna, obsahuje jazyk INTER-PASCAL několik předdefinovaných konstant obsahující možné hodnoty pro vlastnosti pera a štětce.

Při kreslení obrazců do grafického okna se kreslí obrazce čarou, jejíž barva je definována procedurou ImagePenColor. Styl čáry je definován procedurou ImagePenStyle a tloušťka čáry je definována procedurou ImagePenWidth. Plocha nakreslených geometrických obrazců je vyplněna barvou zadanou procedurou ImageBrushColor a stylem zadaným procedurou ImageBrushStyle.

Definice barev

Možnosti zobrazování barev závisí na možnostech videokarty ve Vašem počítači. Používáte-li barevnou VGA kartu, máte možnost zobrazit minimálně 16 barev. Po příslušném nastavení videoadaptéru je možné běžně zobrazovat 256 barev, výjimečně i více. Jazyk INTER-PASCAL umožňuje zadat teoreticky libovolnou barvu z rozsahu 16 miliónů barev. Skutečně zobrazená barva ale závisí na možnostech technického zařízení, protože se zobrazí vždy barva nejbližší.

Hodnotu barev je možné zadávat několika způsoby, které lze v programu libovolně kombinovat. Pokud budete používat pouze základní, šestnáctibarevnou paletu, můžete tak učinit hodnotou barvy v rozsahu 1 až 16. Stejnou barvu máte možnost zadat i pomocí předdefinované konstanty udávající anglické jméno barvy. Budete-li chtít vybírat barvy z rozsahu nad základních šestnáct barev, budete muset použít definici barev pomocí RGB hodnoty.

Hodnota barev zadávaná definicí RGB znamená, že každá barva je definována jako poměr kombinace barev modré, zelené a červené. Pro každou barvu je možné volit hodnoty v rozsahu 0 až 256. Násobek těchto hodnot (modrá x zelená x červená) udává výslednou barvu. Výhodné je používat tzv. hexadecimálního zápisu, kdy jsou pro každou barvu vyhrazeny dvě pozice čísla s hodnotami od 00 (číslo 0) až do FF (číslo 256). Při použití hexadecimálního čísla je nutné uvést před číslem rozlišovací znak $. Viz tabulka hodnot barev.

hodnota	konstanta	název barvy
$000000	clBlack	černá
$000080	clMaroon	kaštanově červená
$0000FF	clRed	světle červená
$008000	clGreen	tmavě zelená
$008080	clOlive	tmavě žlutá
$00FF00	clLime	světle zelená
$00FFFF	clYelow	žlutá
$800000	clNavy	tmavě modrá
$800080	clPurple	tmavě fialová
$808000	clTeal	tmavě modrozelená
$808080	clDkGray	tmavěšedá
$C0C0C0	clLtGray	světle šedá
$FF0000	clBlue	modrá
$FF00FF	clFuschsia	fialová
$FFFF00	clAgua	modrozelená
$FFFFFF	clWhite	bílá

Bílou barvu štětce je proto možné nastavit následujícími příkazy:

```
SetBrushColor($FFFFFF)
SetBrushColor(clWhite)
```

Procedura Arc

```
procedure Arc(x1, y1, x2, y2, x3, y3, x4, y4: integer);
```

Nakreslí kruhový oblouk - část elipsy dle zadaných souřadnic. Souřadnicemi x1, y1, x2, y2 jsou zadány okraje plné elipsy, pokud by byla vykreslená celá. Souřadnicemi x3, y3 je uveden počátek vykreslování křivky a souřadnicemi x4, y4 je udán konec vykreslení křivky proti směru hodinových ručiček.

Příklad:

```
arc(0,0,100,100, 50,0,0,50);         {levý  horní čtvrtkruh}
arc(0,0,100,100, 0,50,50,100);       {levý  dolní čtvrtkruh}
arc(0,0,100,100, 100,50,50,0);       {pravý horní čtvrtkruh}
arc(0,0,100,100, 50,100,100,50);     {pravý dolní čtvrtkruh}

arc(0,0,100,100, 100,50,0,50);       {horní půlkruh}
arc(0,0,100,100, 0,50,100,50);       {dolní půlkruh}
arc(0,0,100,100, 50,100,50,0);       {pravý půlkruh}
arc(0,0,100,100, 50,0,50,100);       {levý  půlkruh}
```

Viz také:
sořadnicový systém

Procedura Ellipse

```
procedure Ellipse(x1, y1, x2, y2 : integer);
```

Procedura nakreslí elipsu, která je umístěna do pomyslného obdélníku o souřadnicích x1, y1 pro levý horní roh a x2, y2 pro pravý dolní roh.

Procedura je vhodná i pro kreslení kružnic, kdy se zadají souřadnice tak, aby tvořily hranu čtverce.

Viz také:
sořadnicový systém

Funkce GetMaxX

```
function GetMaxX : integer;
```

Funkce vrací velikost grafického okna na ose x (šířka okna v bodech).

Viz také:

GetMaxY

Funkce GetMaxY

```
function GetMaxY : integer;
```

Funkce vrací velikost grafického okna na ose y (výška okna v bodech).

Viz také:
GetMaxX

Procedura ImageBrushColor

```
procedure ImageBrushColor(color: word);
```

Procedura nastaví barvu štětce dle definovaných konstant. Při dalším kreslení ploch bude využívána nastavená barva.

Viz také:
definice barev, ImageBrushStyle

Procedura ImageBrushStyle

```
procedure ImageBrushStyle(styl: integer);
```

Procedura nastaví styl vykreslování ploch. Okraje plochy se vykreslují dle definice barvy, tloušťky a stylu pera. Pro zadávání je možné použít předdefinované konstanty:

hodnota	konstanta	název stylu
1	bsSolid	vyplní oblast jednou barvou
2	bsClear	vyplní oblast barvou pozadí
3	bsHorizontal	vyplní oblast vodorovnými čarami
4	bsVertical	vyplní oblast svislými čarami
5	bsFDiagonal	diagonální čáry \\\\\\\
6	bsBDiagonal	diagonální čáry ///////
7	bsCros	vodorovné a svislé čáry
8	bsDiagCross	vodorovné a svislé čáry diagonálně

Viz také:
ImageBrushColor

Procedura ImageClear

```
procedure ImageClear;
```

Provede se výmaz obsahu grafického okna bez změny definice velikosti. Nastaví současně bílou barvu plochy, styl štětce pro plně vybarvené plochy, černou barvu pera, sílu čáry na jeden bod. Pokud požadujete současnou změnu velikosti grafického okna, použijte proceduru ImageInit.

Viz také:
ImageInit

Procedura ImageFontColor

```
procedure ImageFontColor(color: word);
```

Procedura nastavuje barvu fontu pro výstup textů do grafického okna procedurou TextOut. Pro zadání barvy je možné uvést buď číselné vyjádření v hodnotách RGB, nebo lze použít výše uvedené konstanty.

Viz také:
definice barev, ImageFontName, ImageFontSize, ImageFontStyle, ImageSetFont

Procedura ImageFontName

```
procedure ImageFontName(font: string);
```

Procedura nastaví zadaný font jako aktuální pro výstup textů do grafického okna procedurou TextOut. Pokud zadaný font neexistuje, nastaví systém Windows sám přibližný font. Nemění vlastnosti fontu pro formuláře !

Viz také:
ImageFontColor, ImageFontSize, ImageFontStyle, ImageSetFont

Procedura ImageFontSize

```
procedure ImageFontSize(size: integer);
```

Procedura nastaví zadanou velikost v bodech jako aktuální pro výstup textů do grafického okna procedurou TextOut. Nemění vlastnosti fontu pro formuláře !

Viz také:
ImageFontColor, ImageFontName, ImageFontStyle, ImageSetFont

Procedura ImageFontStyle

```
procedure ImageFontStyle(style: integer);
```

Procedura nastaví styl fontu pro výstup textů do grafického okna procedurou TextOut. Je možné nastavit tučné písmo, nakloněné písmo, podtržené a přeškrtnuté písmo. Pro zadávání stylu je možné použít numerickou hodnotu, nebo předdefinované konstanty:

hodnota	konstanta	popis stylu
0	fsNormal	normální písmo
1	fsBold	tučné písmo
2	fsItalic	nakloněné písmo
4	fsUnderline	podtržené písmo
8	fsStrikeOut	přeškrtnuté písmo

Pokud budete chtít nastavit najednou vícenásobný styl fontu, musíte zadat součet hodnot stylu.

Příklad:

Pro nastavení nakloněného podtrženého písma musíte zadat:

```
ImageFontStyle(6);        // (součet 2 + 4) nebo
ImageFontStyle(fsItalic + fsUnderline);
```

Viz také:
ImageFontColor, ImageFontName, ImageFontSize, ImageSetFont

Procedura ImageFromClip

```
procedure ImageFromClip(x1, y1, x2, y2 : integer);
```

Do grafického okna se přesune obsah schránky (clipboardu). Pozice levého horního rohu pro zobrazení je zadána souřadnicemi x1, y1. Pravý dolní roh je zadán souřadnicemi x2, y2.

Příklad:

Pokud budete chtít zobrazit obsah schránky na celou plochu grafického okna, zadejte příkaz:

```
ImageFromClip(0,0,GetMaxX,GetMaxY);
```

Viz také:
ImageToClip

Procedura ImageInit

```
procedure ImageInit(osax, osay: integer);
```

Procedura inicializuje grafické okno a nastaví jeho velikost na rozměry osaX x osaY. Nastaví současně bílou barvu plochy, styl štětce pro plně vybarvené plochy, černou barvu pera, sílu čáry na jeden bod. Inicializací se provede výmaz původního obsahu grafického okna. Při spuštění programu je grafické okno automaticky inicializováno ve velikosti 300 x 400 bodů.

Viz také:
ImageClear

Procedura ImageLoad

```
procedure ImageLoad(files: string);
```

Procedura načte do grafického okna obrázek ze souboru typu *.BMP. Velikost grafického okna bude nastavena podle velikosti obrázku v souboru. Velikost obrázku v bodech je možné zjistit po načtení funkcemi GetMaxX a GetMaxY.

Viz také:
ImageSave

Procedura ImageMove

```
procedure ImageMove(x1, y1, x2, y2, x3, y3, x4, y4: integer);
```

Procedura provádí kopii části plochy grafického okna na zadanou pozici. Ohraničení plochy, která se má kopírovat je zadána souřadnicemi x1, y1, x2, y2. Plocha se zkopíruje na místo zadané souřadnicemi x3, y3, x4, y4. Neodpovídá-li poměr stran zdrojové a cílové plochy, nebude zkopírovaný obraz uříznut, ale bude upraven (deformován) do zadaných souřadnic. To je možné využít k mnoha zajímavým efektům. Například ke zvětšení části plochy apod.

Viz také:
sořadnicový systém

Procedura ImagePenColor

```
procedure ImagePenColor(color: word);
```

Procedura nastavuje barvu pera (čáry) pro kreslení do grafického okna. Pro zadání barvy je možné uvést buď číselné vyjádření v hodnotách RGB, nebo lze použít výše uvedené konstanty.

Viz také:
definice barev

Procedura ImagePenStyle

```
procedure ImagePenStyle(styl: integer);
```

Procedura nastavuje druh čáry, kterou se bude kreslit.

hodnota	konstanta	název stylu
1	psSolid	souvislá čára
2	psDash	přerušovaná čára
3	psDot	tečkovaná čára
4	psDashDot	čerchovaná čára
5	psDashDotDot	čerchovaná čára se dvěma tečkami
6	psClear	neviditelná čára

Procedura ImagePenWidth

```
procedure ImagePenWidth(width: integer);
```

Procedura nastavuje sílu čáry v bodech. Pozor na to, že při použití silnější čáry může styl čáry splynout do plné čáry.

Procedura ImagePrint

```
procedure ImagePrint(x1, y1, x2, y2 : integer);
```

Procedura vytiskne obsah grafického okna na aktuálně nastavenou tiskárnu. Jako parametry se udávají souřadnice v bodech, kam se má obrázek vytisknout. Tím je možné dosáhnout požadovaného roztažení obrazu. Pozor na to, že různé tiskárny mohou mít různý stupeň rozlišení bodů na stránce. V konečné verzi programu bude procedura pravděpodobně ještě změněna !!!

Zadají-li se nulové parametry, bude obrázek vytisknut tak, aby byl rovnoměrně umístěn na stránce papíru. Nedojde přitom k žádné deformaci obrazu.

Procedura ImageSave

```
procedure ImageSave(files: string);
```

Procedura uloží obsah grafického okna na disk do souboru definovaného parametrem files ve formátu *.BMP. Jestliže soubor zadaného jména již existuje, bude přepsán.

Viz také:
ImageLoad

Procedura ImageSetFont

```
procedure ImagesetFont;
```

Procedura vyvolá dialog Windows pro definici fontu. Bude Vám nabídnut seznam instalovaných fontů se současnou možností zadání velikosti, stylu a barvy.

Viz také:
ImageFontColor, ImageFontName, ImageFontSize, ImageFontStyle

Procedura ImageToClip

```
procedure ImageToClip(x1, y1, x2, y2 : integer);
```

Část grafického okna zadaného souřadnicemi levého horního rohu x1, y1 a pravého dolního rohu x2, y2 se přesune do schránky (clipboardu).

Příklad:

Pokud budete chtít přesunout do schránky obsah celého grafického okna, zadejte příkaz:

```
ImageFromClip(0,0,GetMaxX,GetMaxY);
```

Viz také:
ImageFromClip

Procedura Line

```
procedure Line(x1, y1, x2, y2 : integer);
```

Nakreslí čáru z pozice zadané souřadnicemi x1, y1 do pozice zadané souřadnicemi x2, y2. Čára se kreslí aktuálním stylem a aktuální barvou. Grafický ukazatel se nastaví na poslední bod nakreslené čáry.

Příklad:

```
Line(10, 10, 60, 100);
```

Viz také:
sořadnicový systém

Procedura LineTo

```
procedure LineTo(x, y : integer);
```

Nakreslí čáru z pozice aktuálního ukazatele do definovaného bodu. Parametry x a y definují koncový bod čáry. Grafický ukazatel se nastaví na poslední bod kreslené čáry.

Příklad:

```
LineTo(60, 100);
```

Viz také:
sořadnicový systém

Procedura MoveTo

```
procedure MoveTo(x, y: integer);
```

Přemístí grafický ukazatel do definovaného bodu zadaného souřadnicemi x a y.

Příklad:

```
MoveTo(60, 100);
```

Viz také:
sořadnicový systém

Procedura Pie

```
Pie(x1, y1, x2, y2, x3, y3, x4, y4: integer);
```

Nakreslí kruhovou výseč - část plochy elipsy dle zadaných souřadnic. Souřadnicemi x1, y1, x2, y2 jsou zadány okraje plné elipsy, pokud by byla vykreslená celá. Souřadnicemi x3, y3 je uveden počátek vykreslování plochy a souřadnicemi x4, y4 je udán konec vykreslení plochy proti směru hodinových ručiček.

Příklad:

```
Pie(0,0,100,100, 50,0,0,50);     {levý  horní čtvrtkruh}
Pie(0,0,100,100, 0,50,50,100);   {levý  dolní čtvrtkruh}
Pie(0,0,100,100, 100,50,50,0);   {pravý horní čtvrtkruh}
Pie(0,0,100,100, 50,100,100,50);{pravý dolní čtvrtkruh}

Pie(0,0,100,100, 100,50,0,50);   {horní půlkruh}
Pie(0,0,100,100, 0,50,100,50);   {dolní půlkruh}
Pie(0,0,100,100, 50,100,50,0);   {pravý půlkruh}
Pie(0,0,100,100, 50,0,50,100);   {levý  půlkruh}
```

Viz také:
sořadnicový systém

Procedura Point

```
procedure Point(x, y, w : integer);
```

Procedura nakreslí na pozici zadané souřadnicemi x, y bod, jehož velikost je zadaná parametrem w. Bod se nakreslí aktuální barvou pera, kterou lze nastavit procedurou ImagePenColor. Bod není vykreslen kružnicí, ale pravidelným mnohoúhelníkem.

Příklad:

```
Point(100, 100, 2);
```

Viz také:
sořadnicový systémSOURADNICE, ImagePenColor

Procedura Rectangle

```
Rectangle(x1, y1, x2, y2 : integer);
```

Nakreslí pravoúhelník. Parametry x1 a y1 definují souřadnice levého horního rohu ravoúhelníku, parametry x2 a y2 definují souřadnice pravého dolního rohu pravoúhelníku.

Příklad:

```
Rectangle(10, 10, 60, 60);
```

Viz také:
sořadnicový systém

Procedura RoundRect

```
RoundRect(x1, y1, x2, y2 : integer);
```

Nakreslí pravoúhelník se zaoblenými hranami. Parametry x1 a y1 definují souřadnice levého horního rohu pravoúhelníku, parametry x2 a y2 definují souřadnice pravého dolního rohu pravoúhelníku. Parametr x3 udává velikost zakřivení pravoúhelníku na ose x, parametr y3 udává velikost zakřivení pravoúhelníku na ose y.

Příklad:

```
RoundRect(10, 10, 60, 60, 5, 5);
```

Viz také:
sořadnicový systém

Funkce TextHeight

```
function TextHeight(s : string) : integer;
```

Funkce vrací výšku textu v bodech dle nastaveného fontu, pokud by se provedl jeho výstup do grafického okna. Proceduru je možné použít například pro orámování textu čarou.

Viz také:
TextOutTEXTOUT, TextWidth

Procedura TextOut

```
procedure TextOut(x, y: integer; Text: string);
```

Zobrazí v grafickém okně na pozici x, y textový řetězec zadaný v parametru Text. Pro zobrazení se použije aktuálně nastavený font.

Viz také:
TextHeight, TextWidth

Funkce TextWidth

```
function TextWidth(s : string) : integer;
```

Funkce vrací šířku textu v bodech dle nastaveného fontu, pokud by se provedl jeho výstup do grafického okna. Proceduru je možné použít například pro orámování textu čarou.

Viz také:
TextHeight, TextOut

Procedura Triangle

```
procedure Triangle(x1, y1, x2, y2, x3, y3 : integer);
```

Procedura nakreslí trojúhelník. Parametry x1 a y1 definují první vrchol, parametry x2 a y2 definují druhý vrchol a parametry x3 a y3 definují třetí vrchol trojúhelníku.

Viz také:
souřadnicový systém

9.10. KNIHOVNA PRO ZOBRAZENÍ GRAFŮ

V knihovně pro zobrazování grafů jsou definovány procedury a funkce sloužící k zobrazování grafů, zadávání hodnot a nastavení zobrazení grafu.

Chart3	- přepínání zobrazení 2D a 3D grafu
ChartData	- definuje data grafu
ChartGrid	- definuje použití vodorovné a svislé mřížky
ChartHide	- minimalizuje okno s grafem do ikony
ChartInit	- inicializuje graf, definuje počet hodnot
ChartLegend	- definuje legendu ke grafu (popis os)
ChartLoad	- načte graf ze souboru z disku
ChartSave	- uloží graf do souboru na disk
ChartShow	- obnoví původní velikost okna s grafem
ChartTitle	- definuje nadpisy grafu
ChartType	- změna typu grafu
ChartUpdate	- obnoví zobrazení grafu po aktualizaci hodnot

Viz také: Okno graf

Procedura Chart3D

```
procedure Chart3D(typ : boolean);
```

Procedura nastavuje typ zobrazování grafu. Pokud se zadá parametr hodnoty True, bude se zobrazovat třírozměrný graf. Zadáním parametru False se bude zobrazovat graf pouze dvourozměrně.

Procedura ChartData

```
procedure ChartData(x, y : integer, data ; real);
```

Procedura umožňuje zadání dat do grafu. V parametrech x a y se zadávají osy a v parametru d se zadává hodnota grafu. Ve vývojové betaverzi je možné zatím zadávat pouze celá čísla. Číslo obsahující desetinnou část bude zaokrouhleno. V dalších verzích bude možné zadávat i reálná čísla (obsahující desetinnou část).

Procedura ChartGrid

```
procedure ChartGrid(g : integer);
```

V zobrazeném grafu je možné zobrazit mřížku pro snadnější vytyčení hodnot grafu na ose. Lze zobrazit vodorovné čáry, svislé čáry, vodorovné i svislé čáry, případně mřížku vůbec nezobrazovat. Typ mřížky závisí na hodnotě parametru g:

hodnota g	zobrazená mřížka
0	zruší zobrazování mřížky
1	zobrazí vodorovné čáry v mřížce
2	zobrazí svislé čáry v mřížce
3	zobrazí vodorovné i svislé čáry v mřížce

Procedura ChartHide

```
procedure ChartHide
```

Procedura minimalizuje okno s grafem do tvaru ikony. Ukrytím okna nedochází k žádné změně hodnot, ani nastavení grafu.

Procedura ChartInit

```
procedure ChartInit(x, y : integer);
```

Procedura inicializuje okno s grafem a nastavuje základní výchozí nastavení. Parametr x definuje počet možných hodnot na ose x, parametr y definuje počet možných hodnot na ose y. Inicializací se provede současné nulování hodnot. Případná předchozí data grafu budou ztracena.

Procedura ChartLegend

```
procedure ChartLegend(x : integer, s : string);
```

Procedura umožňuje popsat hodnoty na ose x. Jako parametr x se uvádí index na ose x, parametr s udává popis osy.

Příklad:

```
ChartLegend(1,"Leden");
ChartLegend(2,"Únor");
```

Procedura ChartLoad

```
procedure ChartLoad(files : string);
```

Načte graf ze souboru uložený dříve na disku počítače. Soubor má standardně přednastavenou příponu *.CHF. Načítá se současně nastavení typu grafu, popisy a legenda grafu. S grafem je možné dále pracovat, měnit nastavení a pod.

POZOR ! V této vývojové betaverzi nesmíte zatím provést změny hodnot. I po načtení grafu jsou známy pouze vnitřně předchozí hodnoty. V dalších verzích programu bude přepracováno.

Procedura ChartSave

```
procedure ChartSave(files : string);
```

Uloží vytvořený graf do souboru na disk. Soubor má standardně přednastavenou příponu *.CHF. Mimo hodnot grafu se ukládají současně i nastavení typu grafu, popisy a legenda grafu.

Procedura ChartShow

procedure ChartShow

Procedura obnoví velikost okna s grafem do podoby před jeho minimalizací do ikony.

Procedura ChartTitle

```
procedure ChartTitle(n : byte, t : string);
```

Procedura nastavuje texty uvedené na okrajích grafu. Text je možné zobrazit nad graf, pod graf, vpravo od grafu a nalevo od grafu. Parametr n udává umístění textu předávaného v parametru t.

```
0 - levý popis
1 - pravý popis
2 - horní popis
3 - dolní popis
```

Například zápis ChartTitle(2,"NADPIS GRAFU") umístí nadpis nad graf. Ve vývojové betaverzi není zatím možné zadávat velikost a druh fontu. Bude dopracováno v dalších verzích.

Procedura ChartType

```
procedure ChartType(typ : integer);
```

Procedura nastaví typ zobrazovaného grafu. Je možné si vybrat z devíti možných typů grafů. Další variantou je potom možnost nastavení, zda se má graf zobrazovat dvourozměrně, případně třírozměrně. To je možné zadat procedurou Chart3D. Parametr typ ve volání procedury pro změnu typu grafu je možné zadat dle následujících možností:

typ	druh grafu
1	čárový
2	sloupcový
3	křivkový
4	bodový
5	kruhový
6	plošný
7	statistický
8	
9	rozdílový

Procedura ChartUpdate

procedure ChartUpdate

Procedura aktualizuje zobrazení hodnot grafu. Je vhodné ji použít až po změně všech požadovaných hodnot, protože samotná změna hodnoty grafu procedurou ChartData neaktualizuje změnu zobrazení hodnot grafu.

Příklad:

```
FOR n := 1 TO 5
    ChartData(1, n, n*n);   {zadání hodnoty grafu}
ENDFOR
ChartUpdate;                {aktualizace zobrazení hodnot}
```

9.11. KNIHOVNA PRO PRÁCI S TABULKAMI

V knihovně pro práci s tabulkami jsou definovány procedury a funkce sloužící k práci s tabulkami ve formátu Excel verze 4, případně vlastním optimalizovaném formátu.

SpreadAutoRecalc - aktivuje/deaktivuje okamžitý přepočet tabulky
SpreadClear - vymaže obsah tabulky
SpreadCol - aktivuje zadaný sloupec
SpreadFormula - uloží do aktivní buňky zadaný vzorec
SpreadGetMaxCol - vrací počet sloupců tabulky
SpreadGetMaxRow - vrací počet řádků tabulky
SpreadGetNumber - přečte číselnou hodnotu z aktivní buňky
SpreadGetText - přečte textovou hodnotu z aktivní buňky
SpreadHide - minimalizuje okno s tabulkou do ikony
SpreadInit - inicializuje novou tabulku
SpreadLoad - načte zadaný soubor jako tabulku
SpreadNumber - uloží do aktivní buňky zadané číslo
SpreadRecalc - přepočítá tabulku podle zadaných vzorců
SpreadRow - aktivuje zadaný řádek
SpreadSave - uloží tabulku do souboru na disk
SpreadShow - obnoví původní velikost okna s tabulkou
SpreadShowColHead - aktiavace/deaktivace zobrazení záhlaví sloupců
SpreadShowLines - aktivace/deaktivace zobrazení mřížky tabulky
SpreadShowRowHead - aktiavace/deaktivace zobrazení záhlaví řádků
SpreadText - uloží do aktivní buňky zadaný text

Viz také: Okno tabulka

Procedura SpreadAutoRecalc

```
procedure SpreadAutorecalc(rec : boolean);
```

Procedura nastaví vlastnost tabulku, zda se má provádět okamžitě po každé změně hodnot tabulku její přepočítání. Pokud je zadána v parametru rec hodnota True, bude se provádět okamžité přepočítávání hodnot tabulky po každé změně. Zadá-li se však hodnota False, nebude se provádět změna hodnot tabulky. Její přepočet je v tomto případě možno provést nastavením hodnoty na True, případně voláním procedury SpreadRecalc. Při počáteční definici tabulky je vhodné nastavit, že se nebude tabulka přepočítávat. Až po kompletní definici tabulky, před požadovaným zobrazením vypočtených hodnot je vhodné nastavit hodnotu na True.

Procedura SpreadClear

`procedure SpreadClear`

Procedura zruší kompletně v celé tabulce její obsah (vzorce, texty i hodnoty). Zachováno je pouze nastavení velikosti tabulky.

Procedura SpreadCol

```
procedure SpreadCol(c : word);
```

Procedura nastaví zadaný sloupec tabulky v parametru c jako aktivní. Parametr se zadává vždy jako číslo od jedničky. Sloupec A má přitom číslo jedna, sloupec B má číslo dvě atd. Po obdobném zadání řádku tabulky procedurou SpreadRow je možné s vybranou buňkou tabulky dále pracovat (zadat hodnotu, případně vzorec, číst obsah buňky apod.).

Příklad:

```
SpreadCol(2);       {nastaví sloupec B}
SpreadRow(6);       {nastaví řádek číslo šest}
SpreadNumber(123); {do buňku B6 uloží hodnotu 123}
```

Procedura SpreadFormula

```
procedure SpreadFormula(formula : string);
```

Procedura dosadí do aktivní buňku vzorec zadaný v parametru formula. Vzorec se musí zadávat bez počátečního rovnítka, jinak bude hlášena chyba ! Rovnítko ve vzorci je doplněno systémem automaticky.

Příklad:

```
SpreadCol(1);       {nastaví sloupec A}
SpreadRow(3);       {nastaví řádek číslo tři}
SpreadFormula('SUM(A1+A2)');
                    {do buňky A3 uloží vzorec =SUM(A1+A2)}
```

Funkce SpreadGetMaxCol

```
function SpreadGetMaxCol : word;
```

Funkce vrací maximální možný počet sloupců tabulky. Funkci je vhodné použít, pokud chcete provést s každým sloupcem tabulky požadovanou akci. Počáteční počet řádků tabulky se přitom zadává procedurou SpreadInit.

Příklad:

```
FOR n := 1 TO SpreadMaxCol
    ...  {akce prováděné s každým sloupcem tabulky}
ENDFOR
```

Funkce SpreadGetMaxRow

```
function SpreadGetMaxRow : word;
```

Funkce vrací maximální možný počet řádků tabulky. Funkci je vhodné použít, pokud chcete provést s každým řádkem tabulky požadovanou akci. Počáteční počet řádků tabulky se přitom zadává procedurou SpreadInit.

Příklad:

```
FOR n := 1 TO SpreadMaxRow
    ...  {akce prováděné s každým sloupcem tabulky}
ENDFOR
```

Funkce SpreadGetNumber

```
function SpreadGetNumber : real;
```

Funkce přečte číselný obsah aktivní buňky k dalšímu zpracování. Pokud obsahuje buňka vzorec, bude vrácen výsledek výpočtu.

Příklad:

```
SpreadCol(2);              {nastaví sloupec B}
SpreadRow(6);              {nastaví řádek číslo šest}
Writeln(SpreadGetNumber); {vypíše obsah buňky B6}
```

Funkce SpreadGetText

```
function SpreadGetText : string;
```

Funkce přečte obsah aktivní buňky k dalšímu zpracování.

Příklad:

```
SpreadCol(2);              {nastaví sloupec B}
SpreadRow(6);              {nastaví řádek číslo šest}
Writeln(SpreadGetNumber); {vypíše obsah buňky B6}
```

Procedura SpreadHide

procedure SpreadHide

Procedura minimalizuje do tvaru ikony okno s tabulkou. Původní velikost okna se nastaví procedurou SpreadShow.

Procedura SpreadInit

```
procedure SpreadInit(row, col : integer);
```

Procedura inicializuje novou tabulku s prázdným obsahem. Velikost tabulku je nastavena podle parametrů row a col.

Příklad:

```
SpreadInit(12, 30); {inicializuje tabulku 12 sloupců x 30 řádků}
```

Procedura SpreadLoad

```
procedure SpreadLoad(files : string);
```

Procedura načte z disku tabulku. Načítat je možné tabulky formátu Excel verze 4 (přípona souboru *.XLS), případně vlastní optimalizovaný formát s označením přípony *.VTS.

Procedura SpreadNumber

```
procedure SpreadNumber(num : real);
```

Procedura ukládá do aktivní buňky tabulky zadané číslo, které pak může být využito v dalších výpočtech.

Procedura SpreadRecalc

procedure SpreadRecalc

Procedura přepočítá tabulku dle vzorců obsažených v tabulce. Je vhodné použít, pokud nemáte nastaven automatický přepočet tabulky procedurou SpreadAutoRecalc. Nejprve zadáte do tabulky všechna požadovaná data a vzorce a až potom provedete přepočet tabulky.

Procedura SpreadRow

```
procedure SpreadRow(r : word);
```

Procedura nastaví zadaný řádek tabulky v parametru r jako aktivní. Po obdobném zadání sloupce tabulky procedurou SpreadCol je možné s vybranou buňkou tabulky dále pracovat (zadat hodnotu, případně vzorec, číst obsah buňky apod.).

Příklad:

```
SpreadCol(2);       {nastaví sloupec B}
SpreadRow(6);       {nastaví řádek číslo šest}
SpreadNumber(123); {do buňku B6 uloží hodnotu 123}
```

Procedura SpreadSave

```
procedure SpreadSave(files : string);
```

Procedura uloží obsah tabulky do souboru na disk. Vzhledem k tomu, že systém umí pracovat se dvěma formáty tabulek, rozlišuje se jejich formát zadanou příponou souboru. Pro uložení tabulky ve formátu Excel ve verzi 4 použijte příponu *.XLS, pro uložení tabulky ve vnitřním optimalizovaném formátu použijte příponu *.VTS. Můžete přitom tabulku načtenou v jednom formátu uložit ve formátu druhém. Původní tabulka zůstane zachována beze změny.

Procedura SpreadShow

procedure SpreadShow

Procedura obnoví velikost okna s tabulkou do podoby před jeho minimalizací do ikony procedurou SpreadHide.

Procedura SpreadShowColHead

```
procedure SpreadShowColHead(show : boolean);
```

Procedura nastavuje viditelnost záhlaví sloupců tabulky, ve které jsou zobrazovány označení sloupců. Pokud se uvede parametr hodnoty True, bude záhlaví zobrazováno. Při uvedení hodnoty False se nebude záhlaví zobrazovat.

Procedura SpreadShowLines

```
procedure SpreadShowLines(show : boolean);
```

Procedura nastavuje viditelnost čar oddělující mezi sebou jednotlivé buňky tabulky. Pokud se uvede parametr hodnoty True, budou se zobrazovat čáry oddělující buňku tabulky. Při uvedení hodnoty False se nebudou čáry zobrazovat.

Procedura SpreadShowRowHead

```
procedure SpreadShowRowHead(show : boolean);
```

Procedura nastavuje viditelnost záhlaví řádek tabulky, ve kterém jsou zobrazovány čísla řádků. Pokud se uvede parametr hodnoty True, bude záhlaví zobrazováno. Při uvedení hodnoty False se nebude záhlaví zobrazovat.

Procedura SpreadText

```
procedure SpreadText(text : string);
```

Procedura ukládá do aktivní buňky text zadaný v parametru procedury. Pokud je text delší, než šířka buňky, roztáhne se text i do sousedních buněk.

9.12. Knihovna pro práci s databázemi

V knihovně pro práci s databázemi jsou definovány procedury a funkce sloužící k práci s databázemi ve formátu FoxPro.

DbfAppendBlank - přidá prázdný záznam do databáze
DbfAppendFrom - načte (přihraje) data z jiné databáze
DbfBof - test začátku databázového souboru
DbfCopyFile - kopíruje databázi do nového souboru
DbfCopyToInfo - kopíruje strukturu databáze do informační databáze
DbfCopyToText - kopíruje databázi do textového souboru
DbfCopyStru - kopíruje strukturu databáze do nového souboru
DbfContinue - pokračuje v hledání dalšího záznamu(viz DbfLocate)
DbfCount - vrací počet záznamů dle nastaveného filtru
DbfCreateFrom - založí novou databázi z informační databáze
DbfCreateInfo - založí novou prázdnou informační databázi
DbfDelete - označí záznam ke zrušení
DbfDeleted - vrací informaci, zda je záznam určen ke zrušení
DbfEvalLog - vrací výsledek logického databázového výrazu
DbfEvalNum - vrací výsledek numerického databázového výrazu
DbfEvalStr - vrací výsledek řetězcového databázového výrazu
DbfEvalTest - testuje správnost databázového výrazu
DbfEof - test konce databázového souboru
DbfFieldCount - vrací počet položek (sloupců) databáze
DbfFieldDec - vrací počet desetinných míst zadané položky
DbfFieldName - vrací jméno položky dle pořadového čísla
DbfFieldType - vrací typ databázové položky (N/C/L/M)
DbfFieldWidth - vrací délku databázové položky
DbfFound - vrací informaci, zda bylo hledání úspěšné
DbfGo - přesun na zadané číslo záznamu
DbfGoBottom - přesun na konec databázového souboru
DbfGoTop - přesun na začátek databázového souboru
DbfIndexTag - založí nový index databáze
DbfLocate - hledá záznam dle zadané podmínky
DbfPack - fyzicky zruší záznamy označené k výmazu
DbfRecall - obnoví platnost záznamu určeného ke zrušení
DbfRecCount - vrací celkový počet záznamů v databázi
DbfRecNo - vrací číslo aktuálního záznamu
DbfReindex - obnoví indexování databáze
DbfSeek - hledá v databázi dle zadaného klíče
DbfName - vrací název souboru databáze
DbfReadDat - přečte datovou položku z databáze
DbfReadLog - přečte logickou položku z databáze
DbfReadNum - přečte mumerickou položku z databáze
DbfReadStr - přečte řetězcovou položku z databáze
DbfReport - vytiskne sestavu
DbfSelect - aktivace pracovní databázové oblasti
DbfSetDeleted - nastaví, přístupnost zrušených záznamů
DbfSetFilter - nastaví filtr databáze (výběr dle hodnot)
DbfSetOrder - určí, který index bude aktivní
DbfTagArea - dle zadaného jména indexu vrací jeho číslo
DbfTagName - dle zadaného čísla indexu vrací jeho název
DbfSkip - přesun ukazatele v databázovém souboru

DbfUse - otevře databázi
DbfWriteDat - zapíše datovou hodnotu do položky databáze
DbfWriteLog - zapíše logickou hodnotu do položky databáze
DbfWriteNum - zapíše numerickou hodnotu do položky databáze
DbfWriteStr - zapíše řetězcovou hodnotu do položky databáze
DbfZap - fyzicky zruší všechny záznamy databáze

Výrazy xBase se používají v mnoha procedurách a funkcích a jsou zapisovány přímo v syntaxi abázových jazyků typu Dbase, Foxpro a Clipper.

Procedura DbfAppendBlank

```
procedure DbfAppendBlank;
```

Příkaz doplní do otevřené databáze v otevřené datové oblasti prázdný záznam. Ukazatel databáze se nastaví současně na nový záznam.

Funkce DbfAppendFrom

```
function DbfAppendFrom(file: string, typ: integer, filtr: string):boolean;
```

Funkce načte do aktuálního databázového souboru data ze souboru uvedeného v parametru file. Typ načítaných dat a způsob jejich načítání je uveden v parametru typ dle následující tabulky:

```
-------------------------------------------------------------
typ  popis dat
-------------------------------------------------------------
 1   databázový soubor FoxPro
 2   ASCII soubor s oddělením položek čárkami (CDF)
 3   ASCII soubor s pevnou délkou záznamů jako databáze (SDF)
-------------------------------------------------------------
```

Pokud se provádí načtení dat z databázového souboru, budou převedeny pouze data se stejným názvem položek databáze.

V souboru typu CDF (comma date format) jsou jednotlivé datové položky odděleny čárkami. Řetězce jsou přitom uvedeny v uvozovkách. Jeden řádek souboru představuje jednu větu databáze. Řádky mohou mít tudíž rozdílnou délku.

Soubor typu SDF (system date format) je textový soubor, ve kterém jsou položky databáze uloženy vždy se stejnou délkou.

Při používání funkce musíte být opatrní, aby nedošlo ke zničení databáze. Pokud neodpovídá řazení položek databáze načítanému textovému souboru, mohlo by dojít k chybnému načtení položek databáze a tím i k jejímu totálnímu znehodnocení.

Převáděná data mohou být omezena filtrem zadaným podmínkou v parametru filtr. Podmínka filtru je logický výraz zadaný v syntaxi jazyka xBase. Pokud se uvede prázdný řetězec, nebudou načítaná dat žádným způsobem filtrována.

Příklad:

```
IF DbfAppendFrom("C:\DATA\AAA.DBF", 1, "ROK > 1997")
   WTITELN("Načtení proběhlo v pořádku");
ENDIF

DbfAppendFrom("C:\DATA\BBB.TXT", 2, "")
```

Viz také:
DbfCopyFile, DbfCopyToText, výrazy xBase

Funkce DbfBof

```
function DbfBof : boolean
```

Funkce vyhodnocuje, zda je ukazatel databáze nastaven před první záznam databáze. Funkce se používá například velmi často v cyklu pro zpracování každého záznamu databáze směrem k počátku databáze. Pokud není v aktivní databázové oblasti otevřen žádný soubor, je rovněž vrácena hodnota False.

Příklad:

```
WHILE not DbfBof                {dokud není začátek databáze}
   WRITELN(DbfReadStr("NAZ")); {vypíše obsah položky NAZ}
   DbfSkip(-1);                 {jdi na předchozí záznam}
ENDWHILE
```

Viz také:
DbfEof, DbfSkip

Funkce DbfCopyFile

```
function DbfCopyFile(file: string) : boolean;
```

Používá se na kopírování otevřené databáze do jiného databázového souboru. Jméno nové databáze je uvedeno parametrem file. Pokud databáze stejného jména již existuje, bude bez varování nahrazena novou databází. Pokud je rozsah aktuální databáze omezen nastaveným filtrem, budou do nové databáze kopírovány pouze záznamy vyhovující zadanému filtru.

Pokud obsahuje databáze záznamy označené ke zrušení, podléhá kopírování těchto záznamů nastavení přepínače DbfSetDeleted.

Funkce vrací v případě úspěšně provedené kopie logickou hodnotu True.

Viz také:
DbfAppendFrom, DbfCopyToText, DbfCopyStru, DbfSetFilter

Procedura DbfCopyToInfo

```
procedure DbfCopyToInfo(f: string);
```

Procedura založí novou informační databázi obsahující strukturu databáze otevřené v aktuální databázové oblasti. Založená informační databáze obsahuje v každém svém záznamu popis jedné položky databáze, ze které byla založena. Obsah informační databáze můžete potom běžným způsobem editovat, přidávat popis nových položek, případně zrušit položky, které nemají být následně v databázi obsaženy. Po provedení změn informační databáze je možné vytvořit funkcí DbfCreateFrom novou, prázdnou databázi se strukturou dle obsahu informační databáze. Informační databáze může tedy sloužit k vytvoření nové databáze, případně ke změnám struktury databáze příkazy z programu.

Struktura informační databáze:

jméno položky	typ	délka	desetiny
FIELD_NAME	C	10	0
FIELD_TYPE	C	1	0
FIELD_LEN	N	5	0
FIELD_DEC	N	3	0

Viz také:
DbfCreateFrom, DbfCreateInfo

Funkce DbfCopyToText

```
function DbfCopyToText(file: string, typ: integer) : boolean;
```

Kopíruje aktuální databázi do textového ASCII souboru. Jméno souboru je uvedeno parametrem file. Pokud soubor stejného jména již existuje, bude přepsán bez varování novým obsahem. Pokud je rozsah databáze omezen nastaveným filtrem, budou do textového souboru převedeny pouze záznamy vyhovující zadanému filtru. Funkce vrací logickou hodnotu udávající úspěšnost vytvoření textového souboru.

Pokud obsahuje databáze záznamy označené ke zrušení, podléhá kopírování těchto záznamů nastavení přepínače DbfSetDeleted.

Uspořádání dat v textovém souboru je určeno parametrem typ, který udává způsob oddělení jednotlivých položek dle následující tabulky:

typ	způsob přenosu do textového souboru
1	ASCII soubor s oddělením položek čárkami (CDF)
2	ASCII soubor s pevnou délkou záznamů jako databáze (SDF)

Pokud zadáte oddělení jednotlivých položek čárkou (CDF), vznikne textový soubor, ve kterém bude jedna řádka textu představovat jeden záznam z databáze. Položky z věty budou přitom odděleny čárkami, řetězcové položky budou umístěny v uvozovkách. Datum se uvede číselně ve tvaru RRRRMMDD, logická hodnota T (True) nebo F (False). Každá řádka textového souboru může mít proto jinou délku.

Při zadání oddělení typu SDF bude vytvořen textový soubor, kde bude jedna řádka představovat jeden záznam databáze. Položky věty nebudou nijak odděleny, zapíší se za sebe v délce položek dle zadání struktury databáze. Díky tomu budou mít všechny řádky textového souboru stejnou délku rovnou součtu délky jednotlivých položek databáze.

Viz také:
DbfCopyFile, DbfAppendFrom, DbfSetFilter

Funkce DbfCopyStru

```
function DbfCopyStru(file: string) : boolean;
```

Založí novou, prázdnou databázi se strukturou výchozí databáze. Jméno nové databáze se zadává v parametru file. Pokud soubor stejného jména již existuje, bude přepsán bez varování novým obsahem. Funkce vrací logickou hodnotu udávající úspěšnost kopírování struktury databáze.

Viz také: DbfCopyFile

Funkce DbfContinue

```
function DbfContinue : real;
```

Pokračuje v hledání dalšího záznamu zadaného podmínkou ve funkci DbfLocate. Funkce vrací v případě úspěšného nalezení číslo záznamu, jinak vrací nulovou hodnotu. Nenalezení záznamu je také indikováno funkcí DbfEof.

Příklad:

```
DbfLocate("CENA > 100")           {hledá v databázi cenu>100}
WHILE not DbfEof                  {dokud není konec databáze}
   WRITELN(DbfReadStr("NAZEV"));  {vypíše obsah položky NAZEV}
   DbfContinue;                   {hledá další záznam}
ENDWHILE
```

Viz také: DbfLocate

Funkce DbfCount

```
function DbfCount : longint;
```

Funkce vrací počet záznamů databáze, které odpovídají nastavenému filtru. V závislosti na nastavení, zda se mají brát v úvahu záznamy určené ke zrušení započítává i záznamy určené ke zrušení.

Viz také:

DbfDelete, DbfRecCount, DbfSetDeleted, DbfSetFilter

Funkce DbfCreateFrom

```
function DbfCreateFrom(f: string) : boolean;
```

Funkce založí novou, prázdnou databázi se strukturou dle informační databáze, která obsahuje jména položek nové databáze, jejich typy a délku. Jméno nové databáze se zadává parametrem s, funkce vrací logický příznak úspěšnosti založení databáze. Informační databáze obsahující popis položek nové databáze musí být otevřena v aktuální databázové oblasti.

Pokud budete editovat informační databázi, musíte dodržovat daná omezení struktury databáze, to je například typ položek pouze C, N, D, L nebo M. Délka numerické položky smí být maximálně 19 znaků. Počet desetinných míst může být uveden pouze u numerické položky. Položka typu datum musí mít délku 8, memo položka musí mít délku 10, logická položka musí mít délku 1. Pokud bude uveden nepřípustný údaj, nebude databáze založena a funkce DbfCreateFrom vrátí hodnotu False.

Informační databázi je možné založit pro další editaci jako prázdnou databázi procedurou DbfCreateInfo. Další možností je vytvořit informační databázi procedurou DbfCopyToInfo z existující databáze. V takovém případě již bude informační databáze obsahovat popis struktury databáze, ze které byla vytvořena a je možné provést pouze úpravy.

Viz také:
DbfCopyToInfo, DbfCreateInfo

Procedura DbfCreateInfo

```
procedure DbfCreateInfo(f: string);
```

Procedura se používá pro založení nové, prázdné informační databáze se jménem zadaným v parametru f. Založená prázdná informační databáze bude mít následující strukturu:

jméno položky	typ	délka	desetiny
FIELD_NAME	C	10	0
FIELD_TYPE	C	1	0
FIELD_LEN	N	5	0
FIELD_DEC	N	3	0

Po otevření takto vytvořené informační databáze ji můžete naplnit popisem požadované struktury a následným voláním funkce DbfCreateFrom vytvořit novou databázi zadané struktury. Informační databáze může tedy sloužit k vytvoření nové databáze, případně ke změnám struktury databáze příkazy z programu.

Viz také:
DbfCopyToInfo, DbfCreateFrom

Procedura DbfDelete

`procedure DbfDelete;`

Označí aktuální záznam databáze ke zrušení. Zatím nedojde k jeho fyzickému zrušení. V závislosti na tom zda je nastavena dostupnost záznamů určených ke zrušení (DbfSetDeleted) je možné se záznamy určenými ke zrušení dále pracovat.

Záznamy určené ke zrušení jsou v databázovém okně zobrazeny červenou barvou. Obnova záznamu určeného ke zrušení se provede procedurou DbfRecall. Fyzické zrušení označených záznamů z databáze se provede procedurou DbfPack. Informaci o tom, zda je záznam označen ke zrušení získáte použitím funkce DbfDeleted. Kompletní zrušení všech záznamů z databáze se provede funkcí DbfZap.

Viz také:
DbfDeleted, DbfPack, DbfRecall, DbfSetDeleted

Funkce DbfDeleted

```
function DbfDeleted : boolean;
```

Funkce vrací informaci o tom, zda je aktuální záznam databáze označen ke zrušení.

Viz také:
DbfDelete, DbfPack, DbfRecall, DbfSetDeleted

Funkce DbfEvalLog

```
function DbfEvalLog(vyraz: string) : boolean;
```

Funkce vyhodnotí logický výraz zadaný parametrem vyraz přímo v syntaxi jazyků xBase. Ve výrazu lze použít a vyhodnotit přímo i položky databáze. Funkce vrací logickou hodnotu dle výsledku vyhodnocení výrazu. Pokud není v aktuální datové oblasti otevřena žádná databáze, je vrácen vždy výraz False a to bez ohledu na zadaný výraz. Stejně tak je vrácena hodnota False, pokud je zadán chybný logický výraz.

Příklad:

```
IF DbfEvalLog("DELETED()") then
   WRITELN("Záznam je určen ke zrušení")
ELSE
   WRITELN("Záznam není určen ke zrušení")
ENDIF
```

Viz také:
DbfEvalNum, DbfEvalStr, DbfEvalTest, DbfReadLog, výrazy xBase

Funkce DbfEvalNum

```
function  DbfEvalNum(vyraz: string) : real;
```

Funkce vyhodnotí numerický výraz zadaný parametrem vyraz přímo v syntaxi jazyků xBase. Ve výrazu lze použít a vyhodnotit přímo i položky databáze. Funkce vrací numerickou hodnotu dle výsledku vyhodnocení výrazu. Pokud není v aktuální datové oblasti otevřena žádná databáze, je vrácen vždy nulový výsledek a to bez ohledu na zadaný výraz. Stejně tak je vrácena nulová hodnota, pokud je zadán chybný numerický výraz.

Příklad:

```
WRITELN(DbfEvalNum('a+b'));   {vrátí součet položek databáze}
WRITELN(DbfEvalNum('LEN(s)'));{vrátí délku položky databáze s}
```

Viz také:
DbfEvalLog, DbfEvalStr, DbfEvalTest, výrazy xBase

Funkce DbfEvalStr

```
function DbfEvalStr(vyraz: string) : string;
```

Funkce vyhodnotí řetězcový výraz zadaný parametrem vyraz přímo v syntaxi jazyků xBase. Ve výrazu lze použít k vyhodnocení přímo položky databáze. Funkce vrací výsledek typu string dle výsledku vyhodnocení výrazu. Pokud není v aktuální datové oblasti otevřena žádná databáze, je vrácen vždy jako výsledek prázdný řetězec a to bez ohledu na zadaný výraz. Stejně tak je vrácen prázdný řetězec, pokud je zadán chybný výraz.

Příklad:

```
WRITELN(DbfEvalStr('DTOC(DATE())'));   {vrátí aktuální datum}
WRITELN(DbfEvalStr('SUBS(POL,1,12)')); {vrátí začátek položky}
```

Viz také:
DbfEvalLog, DbfEvalNum, DbfEvalTest, DbfReadStr, výrazy xBase

Funkce DbfEvalTest

```
function DbfEvalTest(vyraz: string) : integer;
```

Funkce zkontroluje syntaktickou správnost výrazu zadávaného v jazyce xBase. Výsledek funkce určí správnost výrazu, případně typ výsledku zadaného výrazu dle následující tabulky:

výsledek	typ výrazu jazyka xBase
0	chybný výraz
1	řetězcový výraz
2	numerický výraz
3	logický výraz

Viz také:
DbfEvalLog, DbfEvalNum, DbfEvalStr, výrazy xBase

Funkce DbfEof

```
function DbfEof : boolean;
```

Funkce vyhodnocuje, zda je ukazatel databáze nastaven za poslední záznam databáze. Funkce se používá velmi často v cyklu pro zpracování každého záznamu databáze.

Příklad:

```
WHILE not DbfEof                    {dokud není konec databáze}
   WRITELN(DbfReadStr("NAZEV"));    {vypíše obsah položky NAZEV}
   DbfSkip(1);                      {jdi na další záznam}
ENDWHILE
```

Viz také: DbfBof

Funkce DbfFieldCount

```
function FieldCount : word
```

Funkce vrací počet položek databáze otevřené v aktuální databázové oblasti.

Viz také:
DbfFieldDec, DbfFieldName, DbfFieldType, DbfFieldWidth

Funkce DbfFieldDec

```
function DbfFieldDec(s: string) : word;
```

Funkce vrací počet desetinných míst numerické položky zadaného jména v otevřené aktuální databázové oblasti.

Viz také:
DbfFieldCount, DbfFieldName, DbfFieldType, DbfFieldWidth

Funkce DbfFieldName

```
function DbfFieldName(i: integer) : string;
```

Funkce vrací jméno položky databáze dle zadání jejího pořadí ve struktuře databáze v otevřené aktuální databázové oblasti.

Viz také:
DbfFieldCount, DbfFieldDec, DbfFieldType, DbfFieldWidth

Funkce DbfFieldType

```
function DbfFieldType(s: string) : string;
```

Funkce vrací typ databázové položky zadaného jména v otevřené aktuální databázové oblasti. Typ položky udává jaké údaje může položka obsahovat: C (řetězec), N (číslo), D (datum), L (logická hodnota), M (memo položka).

Viz také:
DbfFieldCount, DbfFieldDec, DbfFieldName, DbfFieldWidth

Funkce DbfFieldWidth

```
function DbfFieldWidth(s: string) : word;
```

Funkce vrací délku databázové položky zadaného jména v otevřené aktuální databázové oblasti.

Viz také:
DbfFieldCount, DbfFieldDec, DbfFieldName, DbfFieldType

Funkce DbfFound

```
function DbfFound : boolean;
```

Vrací informaci o tom, zda bylo poslední hledání záznamu v databázi dle klíče funkcí DbfSeek úspěšné. Pokud ano, vrací hodnotu True. Pokud nebylo hledání úspěšné, případně jestliže nebylo hledání v aktuální datové oblasti provedeno, vrací funkce hodnotu False.

Příklad:

```
DbfSetOrder("NAZ");   {nastaví klíč dle názvu}
DbfSeek("TELEVIZE");  {hledá v databázi dle klíče text TELEVIZE}
IF DbfFound then
   WRITELN("Záznam byl nalezen");
ENDIF
```

Viz také: DbfSeek

Procedura DbfGo

```
procedure DbfGo(rec: longint);
```

Nastaví ukazatel v databázi na určené číslo záznamu. Ignoruje přitom nastavený filtr a nastavení přístupnosti pouze záznamů neoznačených ke zrušení.

Viz také:
DbfGoTop, DbfGoBottom, DbfSetDeleted

Procedura DbfGoBottom

```
procedure DbfGoBottom;
```

Nastaví ukazatel v databázi na poslední záznam. Pokud není nastaven index databáze, bude se jednat o fyzicky poslední záznam dle pořadí zápisu do databáze. Pokud bude nastaven index, bude nastaven poslední záznam dle nastaveného indexu.

Viz také:
DbfGo, DbfGoTop

Procedura DbfGoTop

```
procedure DbfGoTop;
```

Nastaví ukazatel v databázi na první záznam. Pokud není nastaven index databáze, bude se jednat o fyzicky první záznam dle pořadí zápisu do databáze. Pokud bude nastaven index, bude nastaven první záznam dle nastaveného indexu.

Viz také:
DbfGo, DbfGoBottom

Funkce DbfIndexTag

```
function DbfIndexTag(name: string, key: string) : integer;
```

Založí nový index v indexovém souboru se stejným jménem, jako má databáze, s příponou souboru CDX. Tento indexový soubor (pokud existuje) se otevírá společně s databází. V indexovém souboru typu CDX může být uloženo najednou několik indexů pro hledání funkcí DbfSeek. Každý index je při svém vytváření pojmenován a je možné se mezi němi funkcí DbfSetOrder přepínat.

Indexy se používají pro rychlé vyhledávání v databázi, řazení dle velikosti nebo podle abecedy.

Jméno indexu pomocí kterého se lze následně mezi indexy přepínat se zadává parametrem name. Výraz, podle kterého jsou záznamy seřazeny je uveden v parametru key. Výraz pro indexování se zadává přímo jazyce xBase a měl by být řetězcového typu. Pro indexování čísel použijte funkci STR(), pro převod data použijte funkci DTOS(). Uspořádání indexu je vždy vzestupné.

Každý index je dostupný buď svým jménem zadaným při jeho vytváření, případně pořadovým číslem svého vytvoření. Doporučuje se přitom používat raději názvu indexu.

Příklad:

```
DbfIndexTag("NAZ","UPPER(NAZ)");  {nastaví index dle položky NAZ}
DbfIndexTag("DAT","DTOC(DAT)");   {nastaví index dle položky DAT}
DbfIndexTag("XXX","NAZ+STR(CENA)");
```

Viz také:
DbfReindex, DbfTagArea, DbfTagName, výrazy xBase

Funkce DbfLocate

```
function DbfLocate(vyraz: string) : real;
```

Hledá záznam databáze podle zadané podmínky. Funkce se může použít i pro neindexované databáze, protože se hledání provádí sekvenčně pro každý záznam vyhodnocováním zadané podmínky. Výraz pro hledání se zadává parametrem vyraz přímo v jazyce xBase. Hledání se provádí vždy od aktuálního ukazatele pozice v databázi, dokud se nenalezne záznam vyhovující zadané podmínce. Na tento záznam je následně nastaven ukazatel databáze. V případě neúspěchu hledání je ukazatel databáze nastaven za poslední záznam databáze a následné volání funkce DbfEof vrací hodnotu True. Pokud bylo hledání úspěšné, je možné hledat další záznam dle stejné podmínky funkcí DbfContinue.

Funkce vrací v případě úspěšného hledání číslo nalezeného záznamu a následná funkce DbfEof vrací hodnotu False.

Příklad:

```
DbfLocate("UPPER(SUBS(NAZ,1,3))='FAX'"); {složitější hledání}
DbfLocate("CENA > 100")           {hledá v databázi cenu>100}
WHILE not DbfEof                  {dokud není konec databáze}
   WRITELN(DbfReadStr("NAZEV"));  {vypíše obsah položky NAZEV}
   DbfContinue;                   {hledá další záznam}
ENDWHILE
```

Viz také:
DbfContinue, DbfEof, výrazy xBase

Procedura DbfPack

```
procedure DbfPack;
```

Zruší fyzicky z databáze záznamy označené ke zrušení. Současně aktualizuje indexy.

Viz také:
DbfDelete, DbfRecall, DbfZap

Procedura DbfRecall

procedure DbfRecall;

Obnoví platnost záznamu určeného ke zrušení.

Příklad:

```
DbfGoTop                {skok na začátek databáze}
WHILE not DbfEof        {dokud není konec databáze}
   IF DbfDeleted then   {pokud se má záznam zrušit}
      DbfRecall;        {obnoví platnost záznamu}
   ENDIF
   DbfSkip(1);          {přejde na další záznam}
ENDWHILE
```

Viz také:
DbfDelete, DbfDeleted, DbfSetDeleted

Funkce DbfRecCount

```
function DbfRecCount : longint;
```

Vrací fyzický počet záznamů databáze. Nebere ohled na nastavený filtr a záznamy určené ke zrušení.

Viz také:
DbfCount, DbfDelete, DbfRecNo

Funkce DbfRecNo

```
function DbfRecNo : longint;
```

Vrací číslo aktuálního záznamu databáze.

Viz také: DbfRecCount

Procedura DbfReindex

```
procedure DbfReindex;
```

Přeindexuje záznamy aktuální databáze.

Viz také:
DbfIndexTag, DbfSetOrder

Funkce DbfSeek

```
function Dbfseek(key: string) : boolean;
```

Hledá v aktuální databázi zadaný klíč zadaný parametrem key. Databáze musí být indexována podle hledaných položek databáze. Klíč pro hledání musí být zadán v souladu v vytvořeným indexem a může v něm být použit výraz v jazyce xBase.

Pokud je hledání úspěšné, vrací funkce hodnotu True a ukazatel databáze je umístěn na nalezený záznam. Následné volání funkce DbfFound vrátí hodnotu True. V případě neúspěšného hledání je vrácena hodnota False a ukazatel databáze je umístěn na nejbližší následující záznam vzestupně podle nastaveného indexu.

Příklad:

```
DbfSetOrder("NAZ");       {nastaví index podle názvu}
DbfSeek("FAX");           {hledá dle názvu text 'FAX'}
DbfSetOrder("DAT");       {nastaví index podle data}
DbfSeek("DTOS(DAY())");   {hledá v položce DAT dnešní datum}
```

Viz také:
DbfFound, DbfIndexTag, DbfSetOrder

Funkce DbfName

```
function DbfName : string;
```

Vrací jméno otevřeného databázového souboru. Pokud není v aktuální datové oblasti otevřena žádná databáze je vrácen prázdný řetězec.

Viz také: DbfUse

Funkce DbfReadDat

```
function DbfReadDat(field: string) : string;
```

Funkce vrací hodnotu zadané datumové položky databáze. Pokud není v aktuální datové oblasti otevřena žádná databáze, vrací funkce prázdný řetězec. Stejně tak je vrácen prázdný rětězec pokud se zadá jméno neexistující položky databáze, případně pokud zadaná položka není datumového typu.

Příklad:

```
WRITELN(DbfReadDat("DATUM"));
```

Viz také:
DbfEvalStr, DbfReadLog, DbfReadNum, DbfReadStr

Funkce DbfReadLog

```
function DbfReadLog(field: string) : boolean;
```

Funkce vrací hodnotu zadané logické položky databáze. Pokud není v aktuální datové oblasti otevřena žádná databáze, vrací funkce hodnotu False. Stejně tak je vrácena hodnota False pokud se zadá jméno neexistující položky databáze, případně pokud zadaná položka není logického typu.

Příklad:

```
IF DbfReadLog("FAK") then
   WRITELN("Faktura byla vystavena");
ENDIF
```

Viz také:
DbfEvalLog, DbfReadDat, DbfReadNum, DbfReadStr

Funkce DbfReadStr

```
function DbfReadStr(field: string) : string;
```

Funkce vrací hodnotu zadané řetězcové položky databáze. Pokud není v aktuální datové oblasti otevřena žádná databáze, vrací funkce prázdný řetězec. Stejně tak je vrácen prázdný rětězec v pokud se zadá jméno neexistující položky databáze, případně pokud zadaná položka není řetězcového typu.

Příklad:

```
WRITELN(DbfReadStr("NAZ"));
```

Viz také:
DbfEvalStr, DbfReadLog, DbfReadNum, DbfReadDat

Funkce DbfReadNum

```
function DbfReadNum(field: string) : real;
```

Funkce vrací hodnotu zadané numerické položky databáze. Pokud není v aktuální datové oblasti otevřena žádná databáze, vrací funkce nulovou hodnotu. Stejně tak je vrácena nulová hodnota pokud se zadá jméno neexistující položky databáze, případně pokud zadaná položka není numerického typu.

Příklad:

```
WRITELN(DbfReadNum("CENA"):12:2);
```

Viz také:
DbfEvalNum, DbfReadLog, DbfReadDat, DbfReadStr

Procedura DbfReport

```
procedure DbfReport(file: string,  s1: integer,  s2: integer,
                    prev: boolean);
```

Vytiskne tiskovou sestavu dle zadané předlohy zadanou v parametru file. Pomocí parametrů s1 (počáteční strana sestavy) a s2 (konečná strana sestavy) je možné zadat rozsah tisku požadovaných stran sestavy. Pokud se zadají nulové hodnoty, bude vytištěna celá sestava bez omezení. Parametr prev udává, zda se má tisková sestava nejprve zobrazit na obrazovce. Při zadání False bude proveden ihned tisk sestavy.

Předloha tiskové sestavy musí být nejprve vytvořena interaktivně z databázového okna kliknutím na ikonu s tiskárnou. Před tiskem sestavy je vhodné nejprve nastavit filtr výběru dat a indexování databáze.

V demoverzi je možné tisknout pouze první stranu tiskové sestavy.

Příklad:

```
{tisk kompletní sestavy, nejprve prohlížení na obrazovce}
DbfReport("CENIK", 0, 0, True);
{tisk stran 10 až 12 sestavy, ihned na tiskárnu}
DbfReport("CENIK", 10, 12, False);
```

Procedure DbfSelect

```
procedure DbfSelect(x: byte);
```

Aktivuje zadanou pracovní databázovou oblast. Systém OZOGAN KLONDAIK může pro práci s databázemi používat tři datové oblasti, v nichž může mít otevřenou databázi. Při spuštění systému je nastavena vždy první datová oblast.

Příklad:

```
DbfSelect(1);                    {nastaví první datovou oblast}
DbfGoTop                         {skok na začátek databáze}
WHILE not DbfEof                 {dokud není konec databáze}
   IF DbfEvalLog("CEN > 100")    {pokud je cena větší než 100}
      pNaz := DbfReadStr("NAZ");
      pCen := DbfReadNum("CEN");
      DbfSelect(2);              {nastaví druhou datovou oblast}
      DbfAppendBlank;            {přidá nový, prázdný záznam}
      DbfWriteStr("NAZ",pNaz);   {zapíše hodnotu do položky}
      DbfWriteNum("CEN",pCen);   {zapíše hodnotu do položky}
      DbfSelect(1);              {nastaví první datovou oblast}
   ENDIF
   DbfSkip(1);                   {přejde na další záznam}
ENDWHILE
```

Viz také: DbfuUse

Procedura DbfSetDeleted

```
procedure DbfSetDeleted(d: boolean);
```

Určuje, zda budou dostupné a viditelné záznamy určené ke zrušení.

Příklad:

```
DbfSetDeleted(True);
WRITELN(DbfCount);      {počet záznamů bez ohledu na zrušení}
DbfSetDeleted(False);
WRITELN(DbfCount);      {počet pouze platných záznamů}
```

Viz také:
DbfCount, DbfDelete, DbfDeleted, DbfRecall

Procedura DbfSetFilter

```
procedure DbfSetFilter(filtr: string);
```

Nastavuje podmínku, která určuje dostupný výběr (filtraci) přístupných záznamů. Filtr se zadává logickým výrazem v parametru filtr. Výraz musí být uveden v syntaxi jazyka xBase. Pokud je zadán chybný výraz, není nastaven žádný filtr. Pokud chcete zrušit nastavení filtru, zadává se pro výraz filtru prázdný řetězec.

Příklad:

```
{položka CENA je větší než 100 a zároveň menší než 200}
DbfSetFilter("CENA>100.AND.CENA<200");
{datumová položka DAT obsahuje větší, než aktuální datum}
DbfSetFilter("DAT>DATE()");
```

Viz také: DbfCount

Funkce DbfSetOrder

```
function DbfSetOrder(name: string) : integer;
```

Určuje, který index bude pro databázi aktivní. Požadovaný index se zadává jménem indexu. Pokud proběhlo nastavení indexu v pořádku, je vráceno číslo indexu. Pokud nešlo index nastavit (např. chybně zadané jméno indexu), je vrácena nulová hodnota. Pokud se má zrušit nastavení databáze podle indexu, zadejte jako parametr prázdný rětězec.

Příklad:

```
DbfSetOrder("NAZ");    {nastaví index podle názvu}
DbfSeek("TELEVIZE");   {hledá zadaný název}
DbfSetOrder("DAT");    {nastaví index podle data}
DbfSeek("DATE()");     {hledá aktuální datum}
DbfSetOrder("");       {zruší nastavení dle indexu}
```

Viz také:
DbfIndexTag, DbfSeek, DbfTagArea, DbfTagName

Funkce DbfTagArea

```
function DbfTagArea(name: string) : integer;
```

Vrací pořadové číslo indexu aktuální databáze podle jeho názvu. Pokud zadáte místo názvu indexu prázdný řetězec, bude vráceno číslo aktuálního indexu.

Viz také: DbfTagName

Funkce DbfTagName

```
function DbfTagName(x: integer) : string;
```

Vrací jméno indexu aktuální databáze podle jeho pořadového čísla. Pokud zadáte místo pořadového čísla indexu nulu, bude vráceno jméno aktuálního indexu.

Viz také: DbfTagArea

Procedura DbfSkip

```
procedure DbfSkip(rec: longint);
```

Posune ukazatel v databázovém souboru o počet záznamů zadaných parametrem rec směrem ke konci nebo začátku souboru (kladné nebo záporné číslo) aktuální datové oblasti. Pokud není aktivní žádný index, je realizován přesun dle fyzického číslování záznamů. Pokud je aktivní index, je realizován přesun v závislosti na nastaveném indexu.

Příklad:

```
DbfGoTop                          {skok na začátek databáze}
WHILE not DbfEof                  {dokud není konec databáze}
   ...                            {libovolné akce s databází}
   DbfSkip(1);                    {přejde na další záznam}
ENDWHILE
```

Viz také: DbfBof, DbfEof

Funkce DbfUse

```
function DbfUse(file: string) : boolean;
```

Otevře v aktuální datové oblasti zadaný databázový soubor typu *.DBF. Pokud existují k datovému souboru i vytvořené indexy, je současně otevřen ihned i příslušný indexový soubor *.CDX, žádný index však nebude aktivní. Ukazatel záznamu je nastaven na první záznam.

Pokud je zadaný soubor nalezen a jedná se skutečně o databázi typu FoxPro, vrací funkci hodnotu True. V opačném případě je při neúspěšném otevření souboru vrácena hodnota False.

Pokud chcete databázový soubor uzavřít, použijte funkci se zadáním prázdného řetězce místo jména souboru.

Příklad:

```
IF DbfUse("CENIK") then            {pokud je databáze otevřena}
   WHILE not DbfEof                {dokud není konec databáze}
      WRITELN(DbfReadStr("NAZ"))   {vypíše položku NAZ}
      DbfSkip(1);                  {přejde na další záznam}
   ENDWHILE
ELSE
   WRITELN("Databázi se nepodařilo otevřít");
ENDIF
DbfUse("");                        {uzavře databázi}
```

Viz také: DbfSelect

Procedura DbfWriteDat

```
procedure DbfWriteDat(field: string, dat: string);
```

Procedura naplní zadanou datumovou položku aktuálního databázového záznamu obsahem parametru dat. Pokud není v aktuální datové oblasti otevřena žádná databáze, případně zadaná položka neexistuje, nebo není typu datum, nebude provedena žádná akce.

Příklad:

```
DbfWriteDat("DATUM", "12/12/1997");
```

Viz také:
DbfWriteLog, DbfWriteNum, DbfWriteStr

Procedura DbfWriteLog

```
procedure DbfWriteLog(field: string, b: boolean);
```

Procedura naplní zadanou logickou položku aktuálního databázového záznamu obsahem parametru b. Pokud není v aktuální datové oblasti otevřena žádná databáze, případně zadaná položka neexistuje, nebo není logického typu, nebude provedena žádná akce.

Příklad:

```
DbfWriteLog("FAK", False);
```

Viz také:
DbfWriteDat, DbfWriteNum, DbfWriteStr

Procedura DbfWriteNum

```
procedure DbfWriteNum(field: string, r: real);
```

Procedura naplní zadanou numerickou položku aktuálního databázového záznamu obsahem parametru r. Pokud není v aktuální datové oblasti otevřena žádná databáze, případně zadaná položka neexistuje, nebo není numerického typu, nebude provedena žádná akce.

Příklad:

```
DbfWriteNum("CENA", 123);
```

Viz také:
DbfWriteDat, DbfWriteLog, DbfWriteStr

Procedura DbfWriteStr

```
procedure DbfWriteStr(field: string, s: string);
```

Procedura naplní zadanou řetězcovou položku aktuálního databázového záznamu obsahem parametru s. Pokud není v aktuální datové oblasti otevřena žádná databáze, případně zadaná položka neexistuje, nebo není řetězcového typu, nebude provedena žádná akce.

Příklad:

```
DbfWriteStr("NAZ", "Tiskárna");
```

Viz také:
DbfWriteDat, DbfWriteLog, DbfWriteNum

Procedura DbfZap

```
procedure DbfZap;
```

Zruší fyzicky najednou všechny záznamy databáze bez ohledu na to, zda jsou záznamy označeny ke zrušení. Zrušení záznamů databáze již nelze žádným způsobem obnovit.

Viz také:
DbfDelete, DbfPack

VÝRAZY XBASE

ALLTRIM(String)

Vrací řetězec bez počátečních a koncových mezer.

Příklad: ALLTRIM(" KLONDAIK ") vrací: "KLONDAIK"

AT(SearchString, TargetString)

Hledá výskyt podřetězce SearchString v řetězci TargetString. V případě nalezení vrací pozici znaku (číslováno od jedničky) v řetězci, kde zadaný podřetězec začíná.

Příklad: AT("Z", "OZOGAN") vrací: 2

CHR(Val)

Vrací znak, jehož číselný ACSII kód odpovídá zadané hodnotě.

Příklad: CHR(83) vrací: "S"

CTOD(String)

Převádí znakový řetězec na datum typu xBase. Zadaný řetězec musí být přitom zadán ve tvaru data.

Příklad: CTOD("12/12/1997")

DATE()

Vrací aktuální datum dle nastavení systémového data počítače.

DAY(Date)

Vrací den v měsíci (číslo od 1 do 31) ze zadaného data.

Příklad: DAY(DATE())

DELETED()

Vrací logickou hodnotu udávající, zda je aktuální záznam databáze označen ke zrušení.

DTOC(Date)

Převádí datumovou položku xBase na řetězec.

DTOS(Date)

Převádí datumovou položku xBase na řetězec ve tvaru RRRRMMMDD, který je vhodný pro indexování databáze podle data.

IIF(Logical, TrueTResult, FalseResult)

Vrací jednu ze dvou hodnot v závislosti na zadané logické podmínce. Výsledné hodnoty mohou být podle způsobu použití typu řetězec, číslo, datum nebo logická hodnota.

Příklad: IIF(MONTH(DATE())=1,"LEDEN","ÚNOR až PROSINEC")

INDEXKEY()

Vrací název aktuálního indexu.

LEFT(String, Lenght)

Vrací zadaný počet znaků z řetězce zleva.

Příklad: LEFT("OZOGAN", 3) Vrací: "OZO"

LEN(String)

Vrací číslo udávající počet znaků v řetězci.

Příklad: LEN("KLONDAIK") vrací 8

LOWER(String)

Převede všechna písmena v zadaném řetězci na malá písmena.

Příklad: LOWER("KlonDaiK") vrací: "klondaik"

MONTH(Date)

Vrací číslo měsíce (1 až 12) odpovídající zadanému datu.

ORDER()

Vrací pořadové číslo aktuálního indexu.

ORDKEY()

RAT(SearchString, TargetString)

Hledá výskyt podřetězce SearchString v řetězci TargetString zprava. V případě nalezení vrací pozici znaku (číslováno od jedničky) v řetězci, kde zadaný podřetězec začíná.

Příklad: AT("Z", "KLONDAIK") vrací: 8

RECCOUNT()

Vrací počet záznamů v aktuální databázi.

RECNO()

Vrací číslo aktuálního záznamu z aktuální databáze.

RIGHT(String, Length)

Vrací zadaný počet znaků z řetězce zprava.

Příklad: LEFT("KLONDAIK", 3) Vrací: "AIK"

SPACE(Length)

Vrací znakový řetězec sestavený ze zadaného počtu mezer.

Příklad: SPACE(3) Vrací: " "

STR(Number, Length, Decimals)

Převádí číslo Number na řetězec o délce Length obsahující počet desetinných míst dle parametru Decimals.

Příklad: STR(123,7,2) vrací: " 123.00"

STRZERO(Number, Length, Decimals)

Převádí číslo Number na řetězec o délce Length obsahující počet desetinných míst dle parametru Decimals. Místo počátečních mezer jsou uvedeny nuly.

Pøíklad: STRZERO(123,7,2) vrací: "0123.00"

SUBSTR(String, Start, Length)

Vrací podřetězec z retězce String od pozice Start a délce Length.

Příklad: SUBSTR("KLONDAIK", 3,4) vrací: "ONDA"

TIME()

Vrací aktuální systémový čas ve tvaru řetězce "HH:MM:SS"

TRIM(String)

Odstraní z rětězce koncové mezeru zprava.

Příklad: TRIM(" KLONDAIK ") vrací: " KLONDAIK"

UPPER(String)

Převede všechna písmena v zadaném řetězci na velká písmena.

Příklad: UPPER("KlonDaiK") vrací: "KLONDAIK"

VAL(String)

Vrací číselnou hodnotu zadaného řetězce.

Příklad: VAL(" 123abc") vrací: 123

YEAR(Date)

Vrací ze zadaného data rok jako číslo.