OZOGAN KLONDAIK (předběžná demoverze)

1.0. Úvod
- 1.1. ZÁKLADNÍ POJMY

2.0. OKNA SYSTÉMU OZOGAN KLONDAIK
- 2.1. Okno příkazy
- 2.2. Okno program
- 2.3. Okno výstup-grafika
- 2.4. Okno výstup-text
- 2.5. Okno sledování proměnných
- 2.6. Okno text

3.0. REŽIMY PRÁCE SYSTÉMU OZOGAN KLONDAIK
- 3.1. Editační režim
- 3.2. Ladící režim
- 3.3. Režim animace
- 3.4. Provozní režim

4.0. MENU SYSTÉMU OZOGAN KLONDAIK
- 4.1.0. SOUBOR
  - 4.1.1. Nový
  - 4.1.2. Otevřít (Ctrl+O)
  - 4.1.3. Uložit (Ctrl+S)
  - 4.1.4. Uložit pod názvem
  - 4.1.5. Konec
- 4.2.0. ÚPRAVY
  - 4.2.1. Vyjmi (Ctrl+X)
  - 4.2.2. Zkopíruj (Ctrl+C)
  - 4.2.3. Vlož (Ctrl+V)
  - 4.2.4. Vymaž
  - 4.2.5. Hledat (Ctrl+F)
  - 4.2.6. Změnit (Ctrl+L)
- 4.3.0. PROGRAM
  - 4.3.1. Spustit (F9)
  - 4.3.2. Skok přes procedury (F8)
  - 4.3.3 Krokovat po řádcích (F7)
  - 4.3.4. Spustit do pozice kurzoru (F4)
  - 4.3.5. Animace programu (F6)
  - 4.3.6. Ukázat aktuální řádek
  - 4.3.7. Restart programu (Ctrl+F2)
  - 4.3.8. STOP programu
- 4.4.0. OKNA
  - 4.4.1. Za sebou
  - 4.4.2. Vedle sebe
  - 4.4.3. Srovnat ikony
  - 4.4.4. Minimalizovat
  - 4.4.5. Příkazy
  - 4.4.6. Program
  - 4.4.7. Výstup-grafika
  - 4.4.8. Výstup-text
  - 4.4.9. Sledování proměnných
  - 4.4.10. Text
- 4.5.0. POMOC

4.5.1. Help
4.5.2. O programu

5.0. KURZ POUŽÍVÁNÍ PROGRAMU
1.lekce - instalace, spuštění a ukončení programu
2.lekce - základy ovládání programu
3.lekce - práce s příkazovým a textovým výstupním oknem
4.lekce - příkaz WRITELN, matematické výpočty
5.lekce - grafické výstupní okno, grafický editor
6.lekce - ovládání grafického okna z příkazového okna
7.lekce - výstup do grafického okna
8.lekce - první program, procedura main
9.lekce - editace programu
10.lekce - proměnné a jejich typy
11.lekce - deklarace a používání proměnných
12.lekce - podmínky v programu, logické výrazy
13.lekce - programové cykly (FOR, REPEAT, UNTIL)
14.lekce - zadání vstupních hodnot, tisk výsledků
15. lekce - vícenásobné větvení programu
16.lekce - struktura programu
17.lekce - deklarace a používání procedur
18.lekce - deklarace a používání funkcí
19.lekce - lokální a globální proměnné
20.lekce - rekurze v programu
21. lekce - návrh a sestavení programu
22. lekce - ladící a animační režim programu

6.0. JAZYK INTER-PASCAL

7.0. Vlastnosti jazyka INTER-PASCAL
7. 1. Co je to program a jak vzniká
7. 2. Základy zápisu programu
7. 3. Struktura programu
7. 4. Definice typů (záznamů)
7. 5. Deklarace proměnných
7. 6. Typy proměnných
7. 7. Proměnné typu pole
7. 8. Deklarace VAR .. ENDVAR
7. 9. Deklarace GLOBAL .. ENDVAR
7. 0. Procedury a funkce
7.11. Výrazy

8. Příkazy jazyka INTER-PASCAL

9. Knihovny procedur a funkcí
9.1. Systémová knihovna
9.2. Matematická knihovna
9.3. Knihovna pro zpracování řetězců
9.4. Knihovna pro práci s textovými soubory
9.5. Knihovna pro zpracování záznamů
9.6. Knihovna pro práci se soubory a adresáři
9.7. Knihovna pro práci s datem a časem
9.8. Interaktivní knihovna
9.9. Grafická knihovna

Vzhledem k tomu, že není vývoj programu OZOGAN KLONDAIK dosud zcela dokončen, je tato verze

programu zveřejňěna jako předběžná demoverze programu. Činíme tak proto, že by jsme rádi získali zkušenosti od uživatelů ještě před dokončením programu. Uvítáme proto Vaše kritické připomínky k programu, případně i náměty na jeho vylepšení a zdokonalení.

V současné době je téměř dokončen vývoj výkonného jádra programu, to je interpretu. Nyní se hlavně dopracovávávají helpy a dokumentace. Zpracovávají se také další příklady programů v jazyce INTER-PASCAL.

Omezení demoverze:
- délka editovaného programu je maximálně 50 řádků
- činnost systému je po 10 minutách ukončena. Po této době je nutné program OZOGAN KLONDAIK znovu spustit.

Liberec 18.8.1997

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, 4 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.

Okno příkazy
Okno program
Okno výstup-grafika
Okno výstup-text
Okno sledování proměnných
Okno text

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.

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 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.

2.6. Okno text

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
Sledování proměnných
Text

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. 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

4.4.10. Text

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, spuštění a ukončení programu
2.lekce - základy ovládání programu
3.lekce - práce s příkazovým a textovým výstupním oknem
4.lekce - příkaz WRITELN, matematické výpočty
5.lekce - grafické výstupní okno, grafický editor
6.lekce - ovládání grafického okna z příkazového okna
7.lekce - výstup do grafického okna
8.lekce - první program, procedura main
9.lekce - editace programu
10.lekce - proměnné a jejich typy
11.lekce - deklarace a používání proměnných
12.lekce - podmínky v programu, logické výrazy
13.lekce - programové cykly (FOR, REPEAT, UNTIL)
14.lekce - zadání vstupních hodnot, tisk výsledků
15. lekce - vícenásobné větvení programu
16.lekce - struktura programu
17.lekce - deklarace a používání procedur
18.lekce - deklarace a používání funkcí
19.lekce - lokální a globální proměnné
20.lekce - rekurze v programu
21. lekce - návrh a sestavení programu
22. lekce - ladící a animační režim programu

Další lekce obsluhy programu OZOGAN KLONDAIK se pøipravují a budou dodávány s plnou verzí programu. Budou obsahovat zejména popis interaktivní knihovnny umožòující tvorbu vstupních formuláøù, popis práce s textovými soubory, zpùsob práce se záznamy a podobnì.

Liberec 18.8.1997

1.lekce - instalace a spuštìní 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 pøíkazovým oknem

Nejprve si ukážeme, jak se dají velmi jednoduše zadávat systému příkazy bez zápisu programu. Najděte na pracovní ploše okno nazvané příkazy 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 - pøíkaz 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 ImageFontColor 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, 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 - promìnné 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 proměnné. 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 promìnných

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
```

POZOR !!!

V současné verzi nesmíte zadat při vstupu do číselné proměnné nenumerický znak. Pokud tak učiníte, bude výsledkem nulová hodnota. V dalších verzích bude upraveno.

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 vìtvení 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 promìnné

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 program 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 animaèní 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.

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
7. 2. Základy zápisu programu
7. 3. Struktura programu
7. 4. Definice typů (záznamů)
7. 5. Deklarace proměnných
7. 6. Typy proměnných
7. 7. Proměnné typu pole
7. 8. Deklarace VAR .. ENDVAR
7. 9. Deklarace GLOBAL .. ENDVAR
7.10. Procedury a funkce
7.11. Výrazy

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
READLN	- 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 READLN

READLN([soubor,] var-name)[;]

Příkaz READLN 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 Pascal je v jazyce definován i příkaz READ, který však provádí stejnou funkci jako příkaz READLN.

Viz také:
proměnné

Příkaz WRITE

WRITE([Soubor,] Expr-1 [:délka][[,] Expr-2[:Délka][[,] Expr-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.

Expr-1 , Expr-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,] Expr-1 [:délka][[,] Expr-2[:Délka][[,] Expr-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.

Expr-1 , Expr-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
9.2. Matematická knihovna
9.3. Knihovna pro zpracování řetězců
9.4. Knihovna pro práci s textovými soubory
9.5. Knihovna pro zpracování záznamů
9.6. Knihovna pro práci se soubory a adresáři
9.7. Knihovna pro práci s datem a časem
9.8. Interaktivní knihovna
9.9. Grafická knihovna

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 - zatím nefunkční, nepoužívat !
GetKey - zatím nefunkční, nepoužívat !

Změna nastavení oken integrovaného prostředí:

CommandHide - minimalizuje příkazové okno do ikony
CommandShow - obnoví původní velikost příkazového okna
ConsoleClear - zruší obsah výstupního textového okna
ConsoleHide - minimalizuje textové výstupní okno do ikony
ConsolePrint - vytiskne obsah textového výstupního okna
ConsoleSave - iuloží obsah textového výstupního okna do souboru
ConsoleShov - obnoví původní velikost textového výstupního okna
ImageHide - minimalizuje grafické okno do ikony
ImageShow - obnoví původní velikost grafického okna
ProgramHide - minimalizuje okno s programem do ikony
ProgramShow - obnoví původní velikost okna s programem

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 pøes reproduktor poèítaèe. Vhodné pro upozornìní na ukonèení výpoètu, uživatelovu chybu a podobnì.

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 CommandHide

`procedure CommandHide;`

Procedura minimalizuje do tvaru ikony pøíkazové okno. Proceduru je vhodné použít na zaèátku programu pro odstranìní nepoužívaných oken a tím ke zpøehlednìní obrazovky.

Procedura CommandShow

```
procedure CommandShow;
```

Procedura obnoví velikost pøíkazového okna do podoby pøed jeho minimalizací do ikony.

Procedura ConsoleClear

`procedure ConsoleClear;`

Procedura vymaže obsah textového výstupního okna. Proceduru je vhodné použít na zaèátku programu pro odstranìní výstupu pøedchozího programu.

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 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 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.

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(l: longint) : longint;
```

Funkce vrací náhodné číslo z intervalu 0 až 1.

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.

Ord - vrací ASCII hodnotu znaku
Chr - na základě čísla z ASCII tabulky vrací příslušný znak
Copy - vrací zadanou část řetězce
Length - vrací délku řetězce
Insert - vloží podřetězec do řetězce
Delete - vyjme část znaků z řetězce
Pos - vyhledá podřetězec v řetězci
Val - provádí konverzi řetězce do numerické hodnoty
Str - provádí konverzi čísla na řetězec
IntToStr - převádí číslo do tvaru řetězce
StrToInt - převádí řetězec do číselné 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.

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
AddPushButton	- definuje na formuláři příkazové tlačítko
AddStatic	- definuje na formuláři statický text
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 (znevidítelní) editační pole
DlgShowControl	- zobrazí (zviditelní) editační pole
DlgIsControlVisible	- zjišťuje zda je editační položka viditelná

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

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 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

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

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:
sořadnicový systém