DR. DEDINSZKY FERENC

CLIPPER 5.0

LSI OKTATÓKÖZPONT

Josef Havel
Max - Stromeyer - Str. 9
7750 Konstanz

DR. DEDINSZKY FERENC

CLIPPER 5.0

Lektor: Kiss József

LSI OKTATÓKÖZPONT
A MIKROELEKTRONIKA ALKALMAZÁSÁNAK
KULTÚRÁJÁÉRT ALAPÍTVÁNY
BUDAPEST, 1991

Kiadó: LSI Oktatóközpont Alapítvány
Felelős kiadó: dr. Kovács Magda
Fedélterv: Székely Edith
Témafelelős: Sziklai Klára
ISBN 963 576 076 0
Készült: AGRO-PRINT Kft. Gyál, 91-67
Felelős vezető: Tóth Antal

TISZTELT OLVASÓ!

A CLIPPER 5.0 kézikönyvben technikai okok miatt néhány oldal hibásan jelent meg.

117., 137., 147., 156., 160 oldal mellékelve helyesen.

120., 129., 134. oldal duplán szerepel.

Olvasóinktól ezúton kérünk elnézést.

Szerző és a Kiadó

Tartalomjegyzék

Bevezetés	5
Technikai leírás	11
Utasítások	35
Az előfordító (pre-processzor) direktívák	59
Parancsok	69
Függvények	171
A Clipper fordítóprogramja	231
Az RTLINK szerkesztőprogram	238
Szabvány osztályok	251
Függelék (kódtáblázatok)	259
Tárgymutató	264

Bevezetés

Magyarországon a Clipper az egyik legelterjedtebb programozási nyelv. Népszerűségét „ősének" — a dBase nyelvnek — köszönheti, amely az első IBM PC kompatibilis gépekkel érkezett hozzánk. A relációs adatbáziskezelő nyelvek közül ez nyújtotta a legnagyobb rugalmasságot és szabadságot a programozók számára. Az első időkben más, ehhez foghatóan hatékony és könnyen kezelhető nyelv nem is állt rendelkezésre, így nem véletlen, hogy a fejlesztések zöme erre épült. Az interpreteres dBase II, dBase III és dBase III Plus után a fordítóprogramos Clipper megjelenése minőségi előrelépést jelentett, ugyanakkor megtartotta, sőt továbbfejlesztette a dBase adatbáziskezelő nyújtotta összes szolgáltatást. Előnyeit látva (gyorsabb futás, az egyetlen EXE állományban áttehető programrendszer, a rendelkezésre álló adatbáziskezelő verziótól független alkalmazási lehetőség, a segédprogramok, a C, illetve Assembler rutinok beépítésének lehetősége stb.) egyre többen áttértek használatára. Elterjedésében a legnagyobb szerepet azonban az játszotta, hogy a korábbi adatbázisokban és programokban legfeljebb minimális változtatást kellett végrehajtani. Ezt a felülről kompatibilitást a mostani 5. verzióban is megőrizték, illetve a — minimálisnak nevezhető — eltérések lekezelésére is gondoltak, amivel fenntartották a korábbi dBase verziók alatt létrehozott adatbázisok további használatának a lehetőségét is.

A Clipper 1987. őszi verziója talán rekordnak is nevezhető ideig tartotta magát. A jelenlegi — 5.0 verziószámmal ellátott — változat felveszi az egyre jobban terjedő objektumorientált programozási nyelvek által odadobott kesztyűt. A dokumentációban a fejlesztők több helyen is hangsúlyozzák, hogy ez a verzió **még nem objektumorientált nyelv**, de az objektumorientált programozásnak már több elemét alkalmazzák benne. Saját objektumosztályokat nem alakíthatunk ki, de a négy objektumosztály kezelésének lehetősége már ízelítőt ad a nyelv várható továbbfejlődési irányáról. Mint minden új verzió első tagját, várhatóan az 5.0-át is hamarosan követi egy javított 5.1. Ez azonban nem tartalmaz majd megváltozott koncepciót és utasításkészletet, csupán az 5.0 gyakorlati alkalmazása során feltárt hibákat javítják ki benne. A könyv címe ezért szól a Clipper 5. verziójáról, és nem csupán az 5.0-ról.

A Clipper 5. verziója a korábbinál is nagyobb mértékben épít az alacsonyabbszintű programnyelveken (C, Assembler) írt rutinok használatára, sőt saját C és Assembler nyelvű rutingyűjteményt is tartalmaz. A gyakorlottabb programozók számára csemegének ígérkezik, hogy ezek a rutinok, sőt a Clipper parancsok és függvények is a saját igényeknek megfelelően alakíthatók, módosíthatók.

A könyv nem azzal a szándékkal íródott, hogy a Clipper nyelvet eddig nem ismerők tankönyvként használva ebből sajátítsák el ennek a programozási nyelvnek és stílusnak a fogásait, de aki már ismer valamilyen másik programnyelvet, annak bátran ajánlhatjuk. Főként azoknak ajánljuk, akik a korábbi Clipper verziókban már némi gyakorlatot szereztek. Ajánljuk azoknak is, akik az eddig hozzáférhető szakirodalomnál valamivel többet szeretnének tudni a Clipper belső lelkivilágáról, és azoknak is, akik csak egy-egy ritkábban használt parancs vagy függvény paraméterezési lehetőségeire kíváncsiak. Legnagyobb hasznát várhatóan azok veszik, akik számára nem áll rendelkezésre dokumentáció, vagy kevéssé ismerik az angol nyelvet, de szeretnék a Clipper programozási nyelv lehetőségeit minél teljesebb mértékben kihasználni.

Az egyes nyelvi elemek alfabetikusan kereshetők ki, ugyanakkor az utasítások, parancsok és előfordító direktívák leírásánál a szűkebb szintaktikai tudnivalók mellett ismertetésre kerül-

nek a működés belső összefüggései is. Ezek egy része magyarul még nem jelent meg, így nemcsak a kezdőbbek, hanem a haladóbbak számára is újdonságot tartalmazhat. Nem esik szó, vagy csak érintőlegesen foglalkozunk a segédprogramokkal (DBU, RL, DEBUG), főképpen azért, mert kezelésük majdnem teljesen azonos a korábbi verzióhoz szállított azonos segédprogramokkal, amelyek a szerző korábbi „*Clipper segédprogramok I-II.*" című könyveiben részletesen ismertetve lettek. Nem esik szó a PE szövegszerkesztőről, mivel úgy érezzük, hogy nem fogja kiszorítani azokat a szövegszerkesztőket, amelyeket eddig használtunk programjaink megírásához.

A Clipper 5. verzió újdonságai

Előfordító (pre-processzor)

A Clipper 5. verziója egy olyan előfordítóval rendelkezik, amely a fordítóprogram számára tartalmaz kiegészítéseket. Ezek:

- kapcsolódó állományok;
- pszeudo-függvények;
- konstansértelmezések;
- feltételhez kötött fordítás.

Felhasználó által definiálható parancsok

- Egy szabvány állomány (STD.CH) tartalmazza a Clipper 5. verzió összes parancsdefinícióját. A felhasználó tetszése szerint megváltoztathatja ezeket, illetve saját parancskészletet állíthat össze.

Új fordítóprogram

A Clipper 5. verzió fordítóprogramja teljesen új:

- Javított kódoptimizálás, beleértve a konstans kifejezések csoportosítását, a kifejezések rövidítését, és a nem elérhető kódok figyelmen kívül hagyását.
- Új fordítási paraméterek, beleértve a feltételes fordítás ellenőrzésével kapcsolatos opciókat; a szabvány parancskészlet kikapcsolása, vagy helyettesítése, a deklarációk ellenőrzése, és üzenetek küldése a nemdeklarált változókról, az előfordító működésével kapcsolatos információk kinyomtatása.
- Környezeti változó használható a saját fordítási paraméterek állandósításához.

Új szerkesztő

A Clipper 5. verzió egy teljesen új szerkesztőprogrammal, az *RTLink*™-kel rendelkezik, nagy programoknál dinamikus overlay technika alkalmazását teszi lehetővé. Újdonságai:

- a Clipper-fordítás utáni tárgykód dinamikus overlay-je;
- belső és külső dinamikus overlay lehetősége;
- a Clipper-tárgykód bővítéses szerkesztése;
- a könyvtárak és tárgyállományok előszerkesztése;
- futás közbeni kódkiosztás a könyvtárak előszerkesztése alapján;

- a C és Assembler programok statikus overlay-je;
- szintaktikai kompatibilitás a Microsoft Link és a PLINK86-Plus szerkesztőkkel.

Új forráskód nyomkövető (debugger)

A Clipper 5. verzió egy új forráskód nyomkövető programmal, a *The Clipper® Debugger*-rel rendelkezik:

- a megszakítási pontok a programfutás szünetét idézik elő, amikor a futás az illető ponthoz ér;
- az időzítési pontoknál megjelennek a képernyőn a változók és a kifejezések értékei;
- a nyomkövetési pontok a program futásának szünetét idézik elő, ha a megadott változó vagy kifejezés értéke megváltozott;
- a hívási ablak lehetővé teszi az aktív eljárások és programblokkok szemrevételezését;
- a változók menetközben megtekinthetők, illetve értékük módosítható.

Változók

A Clipper 5. verzióban a változók rendszere a modularitás érdekében új osztállyal és deklarálási lehetőséggel bővült:

- **Valódi local és static változók.** A *local* változók csak abban a programban érvényesek, amelyben deklarálva lettek, a *static* változók pedig (ha a főprogramban deklaráltuk) a teljes programrendszerben. A local és static változókról a fordítás ideje alatt referencia készíthető. Ezek alkalmazása lényegesen gyorsabb futási időt biztosít, mint a *public* és *private* változóké.
- **Formális paraméterek.** Ha a Clipper 5. verziójában egy függvényt, vagy egy eljárást kívánunk létrehozni, ennek paramétereit a név után zárójelben fel kell sorolni. Az így megadott paraméterek *local* változóként automatikusan deklarálásra kerülnek.
- **Deklarálások.** Az opcionális deklarálási lehetőség az összes változóosztályra érvényes. A nemdeklarált változók a fordítás ideje alatt automatikusan deklarálódnak.

Kiterjesztett tömbkezelés

- **Tömbhívatkozások.** A Clipper 5. verzió tömbjei könnyen kezelhetők. A tömbök által lefoglalt helyek automatikusan megszünnek, amikor már nincs rájuk hívatkozás.
- **Egymásbaágyazott tömbök.** A tömbelemek lehetnek egy másik tömb hívatkozásai; ilymódon komplex adatszerkezetek hozhatók létre.
- **Többdimenziós tömbök.** Az egymásbaágyazott tömbökkel mátrixmodell és komplex többdimenziós tömbök hozhatók létre. A többdimenziós tömb egy elemét a C vagy a Pascal nyelv szintaktikája szerint lehet címezni.
- **Dinamikus méretezés.** A Clipper 5. verzió tömbjei dinamikusan növelhetők, illetve csökkenthetők.
- **A tömb, mint függvényérték.** A felhasználó által definiált függvények visszaadott értékei tömbök is lehetnek, amelyeket a változók sorozatánál sokkal könnyebb kezelni.
- **Konstansokat tartalmazó tömbök.** A tömbök tartalmazhatnak konstansértékeket is.

Új adattípusok

A Clipper 5. verzió két új adattípussal rendelkezik:

- **Kódblokk**: egy lefordított forráskód, amelyet a Clipper adatként kezel. Hasonló a makróhoz, de annál gyorsabb és hatékonyabb. Más értékekhez hasonlóan a kódblokk értéke átadható egy változónak, vagy paraméterként egy függvénynek, illetve onnan visszaadható. A kódblokk tartalma kiértékelhető, ami azt jelenti, hogy a blokkon belüli kódok kiértékelésre kerülnek, az eredmény pedig visszaadódik. A kódblokk blokkparaméterrel deklarálható, amely a függvények és az eljárások formális paramétereihez hasonló.
- **NIL**: Ez az érték automatikusan átadódik minden olyan változónak (kivéve a public változókat), amelyek kezdeti értékadását nem hajtottuk végre. Ugyanígy, ha egy eljárásban vagy függvényhívás során elmulasztottuk a paraméterek definiálását, ezek helyén automatikusan a NIL jelenik meg.

Új és kibővített funkciójú operátorok

- Egy új in-line operátor (:=), amely lehetővé teszi a deklaráló utasításokban történő azonnal értékadást. Ez a fajta értékadás az összes változóosztály számára — beleértve a mezőváltozókat is — alkalmazható.
- Egy új növelő operátor (++), amelyet a numerikus értékek növelésére lehet használni.
- Egy új csökkentő operátor (--), amelyet a numerikus értékek csökkentésére lehet használni.
- A makró operátor (&) kibővített formában alkalmazható, amellyel mostmár komplex karakteres kifejezés is értelmezhető. A kifejezést zárójelek közé kell tenni, például:

```
&(elsokif + masodkif)
```

- Egy új továbbító operátor, amellyel üzenetek küldhetők a felhasználói interface objektumaihoz.

Statikus függvények és eljárások

A Clipper 5. verzióban a felhasználó által definiált függvények és eljárások STATIC deklarációval rendelkezhetnek. A static függvények vagy eljárások egy másik függvény vagy eljárás számára csak akkor érhetők el, ha ugyanabban a .prg állományban lettek deklarálva.

Tökéletesített paraméterátadás

A Clipper 5. verzióban a felhasználó által definiált függvények vagy eljárások argumentumai egy üres értékkel is helyettesíthetők. Ehhez egyszerűen egy vesszőt kell elhelyezni a hiányzó argumentum helyére:

```
Fuggv(arg1,,arg3)
```

GET objektumok

A Clipper 5. verzióban a GET-objektumok a felhasználó által teljeskörűen ellenőrizhetők:

- többszörös GET-listák hozhatók létre;
- egymásbaágyazott READ-ek, párhuzamos READ-ek, felfüggesztett READ-ek alkalmazhatók;
- a navigációs és szerkesztő billentyűk feletti teljeskörű ellenőrzés lehetséges;
- programozható információlekérés a GET-ekről, help szöveg, vagy más információ kapcsolása a GET-hez, és annak későbbi visszakérdezése;

- lekérdezhető állapotinformációk;
- az állapotinformációk dinamikus módosítása, beleértve a koordinátákat, színeket, képeket stb.

TBROWSE objektumok

A Clipper 5. verzió lehetővé teszi az adatok táblázatos formában történő megjelenítését egy előre definiált TBROWSE objektumosztály felhasználásával:

- többszörös megjelenítés, párhuzamos megjelenítés, felfüggesztett megjelenítés;
- a tömbök és a számított mezők értékének hatékony megjelenítése makrók alkalmazása nélkül;
- a léptetés és az adatvisszaadás teljeskörű ellenőrzése a megjelenítés hatókörének (WHILE alkalmazása) és szűrőfeltételének beállításával;
- a navigációs billentyűk teljeskörű ellenőrzési lehetősége;
- a megjelenítés programból vezérelhető felfrissítése;
- programozható oszlopzárolás;
- programozható színbeállítások, amellyel elkülönített és dinamikus színek specifikálhatók minden megjelenítésre kerülő értékhez;
- programozható információlekérés minden oszlopról, az engedélyezett GET-ekről vagy más információkról, amelyek az oszloppal kapcsolatban állnak;
- lekérdezhető állapotinformációk;
- az állapotinformációk dinamikus módosítása, engedélyezett oszlopok hozzáadása vagy törlése, a koordináták változtatása stb.

Módosított adatbázis-vezérlő

Lehetővé teszi korábbi verziójú Clipper vagy dBase adatbázis-vezérlőkkel létrehozott állományok használatát.

Az Extend System

A Clipper 5. verzió több olyan C, illetve Assembler nyelven írt rutint tartalmaz, amelyek az eddigi verziókban nem szerepeltek.

A PE.EXE programszerkesztő

A Clipper 5. verzió a MEMOEDIT() függvény felhasználásával önálló programszerkesztővel rendelkezik. Lehetőségei korlátozottak, így nem váltja ki a professzionális programszerkesztő rendszereket.

Konfiguráció

A Clipper 5. verzió alkotórészei áttekinthető könyvtárszerkezetben találhatók, amely az installálás során jön létre. Ezeknek a könyvtáraknak az elérési lehetőségét az AUTOEXEC.BAT-ban, a Clipper PATH-ban és a környezeti változók értékeinek beállításával kell biztosítani.

On-line dokumentáció

A Clipper 5. verzió dokumentációja a Norton Guide memóriarezidens program segítségével képernyőn megjeleníthető.

A Clipper 5. verzió installálási igénye

- IBM PC, vagy ezzel 100%-ig kompatibilis számítógép;
- 3.1 vagy ennél későbbi DOS verzió;
- legalább 512 Kbyte RAM;
- legalább egy 5,25"-os, vagy 3,5"-os hajlékonylemez egység;
- a merevlemezen legalább 1,5 Mbyte szabad terület (a minimális konfigurációhoz), legfeljebb 3,5 Mbyte a maximális konfigurációhoz.

Az installálás után létrejövő könyvtárszerkezet

Alkönyvtár	Tartalom
\CLIPPER5	A Clipper 5. verzió főkönyvtára
\BIN	A futtatható (.EXE) és a parancs (.BAT) állományok
\INCLUDE	Kapcsolódó (header — .CH és .H) állományok
\LIB	A könyvtár (.LIB) állományok
\PLL	Az előszerkesztési könyvtár (.PLL) állományai
\SOURCE	A forrásállományok (.prg) főkönyvtára
\DBU	Az adatbázis-karbantartó segédprogram (DBU) forrásprogramjai
\PE	A programszerkesztő (PE) forrásprogramjai
\RL	A jelentés és címkekészítő segédprogram (RL) forrásprogramjai
\SAMPLE	Forrásnyelvű példaprogramok
\NG	A Clipper on-line dokumentációjának állományai

Installálás után javasolt a \BIN és az \NG könyvtárakat az AUTOEXEC.BAT-ban a SET PATH utasítással láncolni, így biztosítható, hogy a futtatható Clipper programok, valamint az on-line dokumentáció tetszőleges helyről elérhető.

Fontos figyelmeztetés:

Az \INCLUDE könyvtár STD.CH állománya tartalmazza a Clipper 5. verzió összes szabvány parancsának definícióját, amelyek tetszés szerint, saját igényeinknek megfelelően módosíthatók. Ezt a módosítást azonban soha ne végezzük direktben! Helyette hozzunk létre tetszőleges néven egy másolatot, és ezen módosítgassunk. Ezt a másolatot a /U paraméterrel kapcsolhatjuk a fordítóprogramhoz, például:

```
CLIPPER Sajatpr /U SAJSTD.CH
```

Ha az STD.CH valamilyen oknál fogva mégiscsak tönkremenne, az 1. installáló lemezről egy egyszerű DOS COPY paranccsal bemásolható.

Technikai leírás

Általános tudnivalók

A könyvben használt jelölések

Jel	Értelmezés
<>	az adatot a felhasználónak kell megadni
()	argumentum-, illetve paraméterlista
[]	a megadás szabadon választható
{}	kódblokk, vagy tömbérték megadása
||	kódblokk argumentumlista
—>	függvény visszaadott értéke
...	az előzőleg beírtak ismétlése, vagy tetszőleges utasítás
,	a listaelemek elválasztása
|	két vagy több elem közül szabadon lehet (egyet) választani
@	makró jelölése
;	folytatássor következik
->	átirányító operátor
*	a kompatibilitás érdekében meghagyott parancsok, függvények

A Clipper program szerkezete

Egy Clipper program az alábbi szerkezeti elemeket tartalmazza:

- deklarációs utasítások;
- utasítások, parancsok;
- függvény-, illetve eljáráshívások;
- előfordító direktívák;
- függvények, eljárások leírása;
- kommentárok.

A programot tetszőleges szövegszerkesztővel .prg kiterjesztésű szövegállományba kell másolni. A fordítóprogram a fölösleges szóközöket és tabulátorjeleket figyelmen kívül hagyja. A „kocsivissza-soremelés" (CR-LF) jeleket az utasítás végének értelmezi. A Clipper 5.0-ban — a korábbi verziókkal szemben — külön csoportba soroljuk a deklarációs utasításokat, a végrehajtható utasításokat és a parancsokat.

Deklarációs utasítások

A deklarációs utasítások közé tartozik a STATIC, LOCAL, PUBLIC, PRIVATE, MEMVAR és FIELD utasítás, amelyek változó- és mezőnevek azonosítójának definiálására, illetve (a MEMVAR és a FIELD kivételével) kezdeti értékük megadására szolgál. A PUBLIC és a PRIVATE utasítás a program tetszőleges helyén kiadható, míg a többi deklarációs utasításnak minden más végrehajtható utasítást megelőzően kell állnia. Egyéb eltéréseiket részletesen az egyes utasítások tárgyalásánál ismertetjük.

Utasítások, parancsok

A korábbi verziók együttesen tárgyalták. A különválasztást az magyarázza, hogy míg a parancsok az STD.CH állományban forrás alakban megtalálhatók — azok igény szerint (természetesen kellő ismeretek birtokában, és elővigyázatossággal) — változtathatók, az utasítások nem.

Függvény-, illetve eljáráshívások

Szintaktikailag azonos szerkezetűek, és azonos funkciót töltenek be. A köztük levő különbségeket a FUNCTION, illetve a PROCEDURE utasítás tárgyalásánál ismertetjük. A hívás szempontjából nem teszünk különbséget a Clipper részét alkotó szabványfüggvények, valamint a felhasználó által megírt saját függvények között.

Előfordító direktívák

A Clipper 5.0 teljesen új része. Feladatuk az, hogy konvertálást végezzenek a forráskódok egyik formájáról egy másikra. A Clipper 5.0 teljes parancskészlete az STD.CH állományban található forráskódú alakot felhasználva a #command és a #translate direktívákkal működik.

Függvények, eljárások leírása

A függvények és eljárások a program egy-egy önálló, összefüggő szerkezeti egységét alkotják, amelyek más program részeként, vagy önálló szövegállományban hozhatók létre.

Kommentárok

A program áttekinthetőségét, működésének későbbi megértését segítő szövegek. A Clipper 5.0-ban több formában alkalmazhatók.

Az operátorok

A Clipper 5.0 a korábbi verziókban is alkalmazott operátorokon (műveleti és relációs jelek) kívül több olyan operátort vezetett be, amelyek a forráskódot rövidebbé teszik. A „szokványos" operátorokat itt külön nem ismertetjük, az egyes változótípusoknál az ott alkalmazható műveletek ismertetésre kerülnek. A bevezetett új operátorok segítségével — ahogy ez a táblázatban is látható — egyidőben hajtható végre egy művelet és az első operandus ennek következtében módosult értékének átadása.

Az új értékadó operátorok és hatásuk

Operátor	Forma	Hatás
+=	a+=b	a:=(a+b)
-=	a-=b	a:=(a-b)
*=	a*=b	a:=(a*b)
/=	a/=b	a:=(a/b)
%=	a%=b	a:=(a%b)
**= vagy ^=	a^=b	a:=(a^b)

A relációs operátorok között újként jelent meg az *azonosság* vizsgálatára szolgáló ==. Ez a karakteres összehasonlításoknál jelent a korábbi lehetőségekhez képest újat: értéke csak akkor lesz igaz, ha a két karaktersorozat teljesen azonos (bővebb információk a karakteres változók ismertetésénél találhatók).

Az operátorok alkalmazási lehetőségei az egyes változótípusoknál:

Operátor	Alkalmazható adattípus
=	összes
:=	összes
+=	karakteres, dátum, memo, numerikus
-=	karakteres, dátum, memo, numerikus
*=	numerikus
/=	numerikus
**= vagy ^=	numerikus
%=	numerikus

A += és a -= operátorok dátum típushoz történő alkalmazása esetén az első operandusnak dátum típusúnak, a másodiknak numerikusnak kell lennie. Az eredményt dátum típusúként adja.

A változók

A Clipper program változói többféle szempont szerint is csoportosíthatók, például

- hatókörük;
- típusuk

szerint.

A változók hatóköre

Hatókörüket a deklaráló utasítással határozhatjuk meg:

- **LOCAL**: érvényességük arra az eljárásra, függvényre terjed ki, amelyben deklarálva lettek. Ha ez a függvény vagy eljárás többször is meghívásra kerül, minden alkalommal új LOCAL változók jönnek létre, még akkor is, ha rekurzív módon a függvény önmagát hívja. A LOCAL változók a deklaráció utáni első hivatkozástól kezdve élnek.
- **STATIC**: érvényességük a deklarálás helye szerinti aktuális eljárásra, vagy függvényre, illetve az ebből meghívott összes eljárásra és függvényre kiterjed. Ha a STATIC változót a főprogramban deklaráljuk, akkor a teljes programrendszerben érvényes. Ha deklarálására egy alsóbb szintű eljárásban, vagy függvényben került sor, akkor a felsőbb szintre történő visszatéréssel a változó elvész, kivéve ha a fordításhoz megadjuk a /N paramétert. A STATIC változók deklarálásukkal azonnal létrejönnek, és ha nem adunk nekik azonnal értéket, akkor a NIL értéket veszik fel.
- **PRIVATE**: a futás ideje alatt dinamikusan jönnek létre a PRIVATE vagy a PARAMETERS utasítás hatására. Ha egy változót nem deklarálunk, akkor automatikusan a PRIVATE deklarációt kapja. Érvényességük a deklaráció helyétől mindaddig terjed, amíg egy másik PRIVATE utasítással azonos nevű változót nem deklarálunk, vagy a RELEASE paranccsal nem töröljük. Kezdeti értékadás hiányában a NIL értéket veszik fel.
- **PUBLIC**: dinamikusan a futás ideje alatt jönnek létre. Érvényességük a deklaráció helyétől mindaddig terjed, amíg egy másik PUBLIC utasítással azonos nevű változót nem deklarálunk, vagy egy RELEASE paranccsal nem töröljük. Ha a program valamely részén a PUBLIC változóval azonos nevű PRIVATE változót deklarálunk, akkor a PUBLIC változó érvényessége felfüggesztődik arra az időre, amíg a

PRIVATE változó érvényben van. A deklarált, de kezdeti értékkel el nem látott PUBLIC változók alapértelmezése a *hamis* (.F.) érték.

Részletesebb információk az egyes utasítások ismertetésénél találhatók.

A változók típusai

A Clipper programnyelv az alábbi változótípusokat kezeli:

- dátum;
- karakteres;
- kódblokk;
- logikai;
- memo;
- NIL;
- numerikus;
- tömb.

Dátum típusú változók

A dátum típusú változókkal egy naptári dátum azonosítható. A Clipperben a dátumokkal többféle művelet is végezhető. A dátumbeli elválasztójelek a SET DATE paranccsal állíthatók be, illetve változtathatók (az alapértelmezés az amerikai dátumforma: hh/nn/[cc]ee, ahol hh 1-12, nn 1-31, cc (az évszázad) 0-29, ee értéke 0-99 közé eshet, az elválasztójel a / jel). A dátumok sorbarendezése mindig kronologikus, függetlenül attól, hogy melyik dátumformát alkalmaztuk. A Clipper 5.0-ban a SET EPOCH paranccsal változtatható az évszázadok alapértelmezése.

A Clipper minden helyes dátumot elfogad 0100.01.01. és 2999.12.31. között, illetve értelmezi az üres dátumkifejezést is.

A dátumtípusú változókkal történő számolásokhoz a dátumkifejezést numerikus formára kell konvertálni a CTOD() függvénnyel. A zárójelen belül a dátumot idézőjelek, aposztrófok vagy szögletes zárójelek közé kell tenni:

```
CTOD("86/5/30")
CTOD('86/5/30')
CTOD([86/5/30])
CTOD("1986/05/30")
```

A CTOD csak helyes dátumot konvertál, például a 89/11/31, vagy a 90/02/29 dátumot egyaránt hibásnak veszi, mivel a november csak 30 napos, illetve 1990-ben a február csak 28 napos volt. Hibás dátum esetén üres dátumkifejezést ad vissza.

Fontos! Ha üres dátumkifejezést akarunk konvertálni, akkor a CTOD("") formát alkalmazzuk!

A dátumműveletek

Művelet	Hatása
+	szám hozzáadása egy dátumhoz
-	szám kivonása dátumból
++	dátum növelése
--	dátum csökkentése
= vagy ==	egyezőség vizsgálata
!=,<> vagy #	a nem egyezőség vizsgálata
<	a korábbi dátum teljesülésének vizsgálata
<=	a korábbi vagy megegyező dátum vizsgálata
>	a későbbi dátum teljesülésének vizsgálata
>=	a későbbi vagy megegyező dátum vizsgálata
= vagy STORE	értékadás
:=	azonnali (in-line) értékadás
+=	azonnali hozzáadás és értékadás
-=	azonnali kivonás és értékadás
REPLACE	mezőérték átírása
CDOW()	a hét napjának szöveges meghatározása (angolul)
CMONTH()	a hónap szöveges meghatározása (angolul)
DAY()	a nap sorszámának meghatározása
DESCEND()	konvertálás komplemens formára
DOW()	a nap sorszáma a héten belül
DTOC()	a dátum karakteresre konvertálása SET DATE alakban
DTOS()	a dátum karakteresre konvertálása rendezési alakban
EMPTY()	üres érték vizsgálata
MONTH()	a hónap sorszámának neghatározása
TRANSFORM()	a formázott karaktersorozat konvertálása
TYPE()	az adattípus meghatározása makrók segítségével
VALTYPE()	az adattípus meghatározása direkt módon
YEAR()	az év meghatározása (beleértve az évszázadot is)

Karakteres változók

A fix hosszúságú értékek tárolására a karakteres változókat alkalmazzuk. A változók egyes karakterei a CHR(32)-CHR(126), valamint a CHR(128)-CHR(255), illetve a CHR(0) értéket tartalmazhatják. A karakteres változók hossza nullától 65535 (64K) karakter hosszig terjedhet. A karakteres változók értékét aposztrófok, idézőjelek vagy szögletes zárójelek között kell megadni. A karakteres változó nulla értéke a két határolójel egymás utáni megadása, például "", '', [] egyaránt üres karakteres változót jelent. Ha egy karakteres érték részeként kívánjuk valamelyik határolójelet alkalmazni, akkor a lehetséges határolójelek kombinációját kell alkalmazni, például:

```
szov = "Azt mondta 'nem érdekel'"
```

A karakteres műveletek

Művelet	Hatás
+	összekapcsolás
-	összekapcsolás szóközök nélkül
=	összehasonlítás egyezőség szerint
==	összehasonlítás a teljes azonosság szerint
!=, <> vagy #	összehasonlítás a nem egyezés szerint
<	összehasonlítás a kisebb érték szerint
<=	a kisebb vagy egyenlő érték
>	összehasonlítás a nagyobb érték szerint
>=	a nagyobb vagy egyenlő érték
= vagy STORE	értékadás
:=	azonnali (in-line) értékadás
+=	azonnali összekapcsolás (+) és értékadás
-=	azonnali összekapcsolás (-) és értékadás
$	részkaraktersorozat keresése (mi-miben)
REPLACE	mezőérték átírása
ALLTRIM()	szóközök levágása elölről és hátulról
ASC()	az ASCII kód meghatározása
AT()	karaktersorozat helyének megtalálása
CTOD()	dátumtípusra konvertálás
DESCEND()	komplemens értékre konvertálás
EMPTY()	üres érték vizsgálata
ISALPHA()	kezdőbetű vizsgálata
ISDIGIT()	kezdőszám vizsgálata
ISLOWER()	kezdő kisbetű vizsgálata
ISUPPER()	kezdő nagybetű vizsgálata
LEFT()	a karaktersorozat bal oldali része
LEN()	a karaktersorozat hossza
LOWER()	a betűk kisbetűre konvertálása
LTRIM()	a bevezető szóközök eltüntetése
PADC()	középre igazítás
PADL()	jobbra igazítás (feltöltés balról)
PADR()	balra igazítás (feltöltés jobbról)
RAT()	karaktersorozat helye jobbról
REPLICATE()	ismétlés megadott számszor
RIGHT()	a karaktersorozat jobb oldali része
RTRIM()	a jobb oldali szóközök eltüntetése
SOUNDEX()	hangzás szerinti egyenértékesre konvertálás
SPACE()	üres karaktersorozat létrehozása
STRTRAN()	karaktersorozat részének helyettesítése
STUFF()	karaktersorozat bővítése, illetve részének törlése
SUBSTR()	karaktersorozat részének kiemelése
TRANSFORM()	a formázott karaktersorozat konvertálása
TYPE()	az adattípus meghatározása makrók segítségével

UPPER()	nagybetűre konvertálás
VAL()	numerikus típusra konvertálás
VALTYPE()	az adattípus meghatározása direkt módon

A karakteres értékek összehasonlítása ASCII kódjuk szerint történik. Ha az EXACT ON, akkor két karakteres érték vizsgálata során (A=B)

igaz (.T.) értéket ad vissza, ha:

- B egy üres karaktersorozat
- a karakterenkénti összehasonlítás során B minden karaktere szerepel A-ban;

hamis (.F.) értéket ad vissza, ha:

- B hossza — LEN(B) — nagyobb A hosszánál;
- a karakterenkénti összehasonlítás során eltérés tapasztalható.

SET EXACT ON esetén két karaktersorozat csak akkor egyezik meg, ha teljesen azonosak (a jobb oldali szóközöket nem számítva).

A kódblokkok

A Clipper 5.0 egy teljesen új - adattípusok közé sorolható - eleme. Lényegében a futtatható program egy részének áthelyezése egy másik helyre. Felfogható úgy is, mint egy névnélküli függvényértékadás, mivel a Clipper a kódblokkot értékként kezeli.

A kódblokkok igen hasonlítanak a makrókhoz, de lényeges különbség van köztük. A makrók olyan karaktersorozatok, amelyek a futás ideje alatt jönnek létre, míg a kódblokkok a fordítás ideje alatt. A kódblokkok jóval rugalmasabbak, és hatékonyabbak, mint a makrók, hiszen például makróval nem lehet *static* vagy *local* változó értékét, illetve paraméterértéket továbbítani.

A kódblokk definiálása

{ | [<argumentumlista>] | <kifejezéslista> }

mind az <argumentumlista>, mind a <kifejezéslista> vesszővel elválasztott több értéket is tartalmazhat. A kódblokk argumentumlistájának funkciója, használata és szabályai megegyeznek a függvények és eljárások argumetumlistájánál leírtakkal. A kifejezéseknek helyes Clipper kifejezéseknek kell lenniük, és nem tartalmazhatnak deklarációs vagy ciklus utasításokat.

Fontos! A kódblokkot közrezáró függőleges vonalból mindíg egy párnak kell lennie, ellenkező esetben a Clipper a kapcsos zárójelek között megadottakat tömbnek adott értékadásnak tekinti.

Példák kódblokk megadásokra:

```
{ || "ez egy karaktersorozat" }
{ |p| p + 1 }
{ |x, y| SQRT(x) + SQRT(y) }
{ |a, b, c| Fuggv1(a) , Fuggv2(b) , Fuggv3(c) }
```

Kódblokk műveletek

Művelet	Hatása
=	értékadás
:=	azonnali (in-line) értékadás
AEVAL()	tömbelemek blokkba másolása
DBEVAL()	munkaterület rekordjainak blokkba másolása
EVAL()	értékmeghatározás kódblokk segítségével

A kódblokkok lényegében egysoros függvénynek is felfoghatók. Használatuk az EVAL() függvénnyel lehetséges, ebben két paramétert kell megadni: az első a blokk neve, a második a bemeneti érték. A visszaadott érték a kódblokk szerinti érték. Például:

```
kodblokk := { |ertek| ertek + 17 }
? EVAL(kodblokk, 13)          // Eredmény: 30
```

Az AEVAL() és a DBEVAL() függvények szintén alkalmasak kódblokkal végzendő műveletekre.

A kódblokkok hatóköre arra a függvényre vagy eljárásra terjed ki, amelyben azt létrehozták. Ugyanakkor, mint bármely más adattípus, visszadott értékként átadható egy korábbi eljárásnak. Ezzel a módszerrel a kódblokkon belül esetlegesen használt *local* vagy *static* változók exportálását érjük el a kódblokkon keresztül, például:

```
FUNCTION Egy()
   LOCAL valt1
   valt1 := 17
   blokk := { |szam| szam + valt1 }
   //
   Ketto(blokk)
   //
   RETURN NIL


FUNCTION Ketto( blokk )
   RETURN (EVAL(blokk, 100))      // visszaadva: 117
```

A Ketto függvényben a *valt1* LOCAL változó értékét ezzel a módszerrel fel tudtuk használni; ezt az értéket közvetlenül más módon nem lehetett volna átadni.

A kódblokk olyan speciális adattípus, amely végrehajtható programkódot tartalmaz. Kiválóan alkalmazható olyan „*fekete doboz*" függvények esetében, amelyeknél nem tudjuk a visszaadott érték típusát meghatározni, például az alábbi függvény hibát jelez, ha a felsoroltaktól eltérő (például dátum) adattípussal találkozik:

```
FUNCTION Adattip( adatnez )
   // "Az adatnez karaktersorozatként kinyomtatása"
   DO CASE
   CASE VALTYPE(adatnez) = "C"
      @ 10, 10 SAY adatnez                  // karakteres
   CASE VALTYPE(adatnez) = "N"
      @ 10, 10 SAY LTRIM(STR(adatnez))      // numerikus
   CASE VALTYPE(adatnez) = "L"              // logikai
      @ 10, 10 SAY IF(adatnez, "igaz", "hamis")
```

```
   ENDCASE
RETURN (.T.)
```

A fenti példa szerinti adattípusok az alábbi kódblokkokba irányíthatók:

```
karkod := { |adat| adat }
numkod := { |adat| LTRIM(STR(adat)) }
logkod := { |adat| IF (adat, "igaz", "hamis") }
datkod := { |adat| DTOC(adat) }
```

A kódblokkok alkalmazásával az előző Adattip nevű függvény az alábbi formát veszi fel:

```
FUNCTION Adattip( adatnez, blokk )
   // "Az adatnez karaktersorozatként kinyomtatása"
   @ 10, 10 SAY EVAL(blokk, adatnez)
   RETURN (.T.)
```

A kódblokkok alkalmazására szemléletes példát láthatunk az STD.CH állományban a SET KEY és a SET FUNCTION parancsok definiálásánál.

A kódblokkok nem írhatók át direkt módon adatbázisba, illetve .mem állományba. Ha egy kódblokkot adatbázisba kívánunk helyezni, hogy később fel tudjuk használni az értékét, akkor a következő módon járhatunk el:

```
USE Sue NEW
//
/* "Egy karakteres mezö feltöltése a kódblokkot hordozó karak-
tersorozattal" */
APPEND BLANK
REPLACE blokkmezo WITH "{|| Ertek()}"
.
.
.
// "A blokk felhasználása egy késöbbi alkalommal"
sajblokk := &(blokkmezo)
EVAL(sajblokk)
```

Figyelem! A fenti példában nem alkalmazhattuk egy lépésben az EVAL(&(blokkmezo)) formát, mivel a fordító egyszerre csak egy fordítási lépést hajt végre, ezért ez nem vezetne helyes eredményre.

Fontos! A makrók alkalmazását csak a karakteres kifejezéseknél javasoljuk. Minden más esetben célszerű előnyben részesíteni a kódblokkokat!

A logikai típusú adatok

Logikai típusú az az adat, amely egy Boole-algebra szerinti értéket hordoz, ami lehet: igaz, hamis; igen, nem vagy on, off. Ezeknek megfelelően az értékek valamelyike lehet: *igaz* - y, Y, t, T; vagy *hamis* - n, N, f, F.

A logikai műveletek

Művelet	Hatása
.AND.	a logikai „és" művelete
.OR.	a logikai „vagy" művelete
.NOT. vagy !	a negálás művelete
= vagy ==	egyenlőség vizsgálata
!=, <> vagy #	nemegyenlőség vizsgálata

Rendezéskor a hamis (.F.) érték mindig kisebb, mint az igaz (.T.) érték.

A memo változók

A memo változók lényegében változó hosszúságú karakteres adatok. A memo mezők az adatbázishoz kapcsolódnak, abból 10 karaktert foglalnak el. Ezt pointerként használják az aktuális adat eléréséhez, amely egy .dbt állományban helyezkedik el. Mivel a memo mezők lényegében karakteres mezők, így rájuk a karakteres mezőkre vonatkozó előírások érvényesek, például hosszuk 65535 karakter lehet, két memo mező összehasonlítása is a karakteres típusnál ismertetett módon megy végbe.

Mivel memo típusú csak mező lehet, ezért karaktersorozatos megfelelője nincs. Ugyanakkor karaktersorozat, mint érték megadható egy REPLACE paranccsal a memo mező számára. A következő táblázat tartalmazza azokat a függvényeket, amelyek memo mezőkön értelmezhetők:

Memo műveletek

Művelet	Hatása
HARDCR()	a lágy „kocsivissza" (CR) jel keményre cseréje
MEMOEDIT()	tartalom szerkesztése
MEMOLINE()	a szöveg egy sorának kiemelése
MEMOREAD()	olvasás egy lemezállományból
MEMOTRAN()	a CR-jelek kihagyása
MEMOWRIT()	lemezre írás
MLCOUNT()	sorok számlálása
MLPOS()	a sorpozició meghatározása

Ezek mellett a speciális memo műveletek mellett az összes karakteres művelet is alkalmazható.

A NIL adattípus

A NIL az 5.0 verzióban bevezetett új adattípus. Szükségességét az indokolta, hogy a nemdeklarált adatok ne vezessenek hibajelzéshez. Tetszőleges adattípus esetében alkalmazható, formája: NIL (ami egyben azt is jelenti, hogy a NIL a Clipper 5.0 foglalt kulcsszava). A következő táblázat tartalmazza azokat a műveleteket, amelyek a NIL típussal végezhetők. Nem végezhető mezőmódosítás a REPLACE parancsnak a NIL értéket megadva, de például összekapcsolható a NIL egy karaktersorozattal.

NIL műveletek

Művelet	Hatása
= vagy ==	egyezőség vizsgálata
!=, <> vagy #	nemegyezőség vizsgálata
<	összehasonlítás a kisebb érték szerint
<=	a kisebb vagy egyenlő érték
>	összehasonlítás a nagyobb érték szerint
>=	a nagyobb vagy egyenlő érték
= vagy STORE	értékadás
:=	azonnali (in-line) értékadás egy nem-mező változónak
EMPTY()	a NIL érték vizsgálata
TYPE()	az adattípus meghatározása makróval
VALTYPE()	az adattípus meghatározása direkt módon

Rendezésnél a NIL a létező legkisebb érték.

A táblázatban feltüntetett deklaráló utasítások automatikusan a NIL értéket rendelik a változóhoz, ha nem történt a deklarálással egyidejűleg kezdeti értékadás.

A NIL deklarációs értékadásai

Művelet	Hatása
DECLARE	a NIL érték adása tömbelemeknek
LOCAL	a NIL érték adása local változóknak és tömbelemeknek
PARAMETERS	a NIL érték adása paramétereknek
PRIVATE	a NIL érték adása private változóknak és tömbelemeknek
PUBLIC	a NIL érték adása public tömbelemeknek
STATIC	a NIL érték adása static változóknak és tömbelemeknek

A PUBLIC és a FIELD kivételével minden változó a NIL értéket kapja kezdőértékként, ha más kezdeti értékadást nem adunk meg. A PUBLIC kezdeti értéke a hamis (.F.) érték, míg a FIELD mező nem tartalmazhat NIL értéket. Függvény- vagy eljáráshíváskor azok a paraméterek, amelyek számára nem lett érték átadva, szintén a NIL értéket veszik fel.

Numerikus adattípus

Numerikus típusúnak azokat az adatokat definiáljuk, amelyekkel matematikai műveleteket (szorzás, osztás stb.) kívánunk végezni. A Clipper a numerikus adattípusban a számjegyeket (0-tól 9-ig), a tizedes pontot, valamint a plusz és a minusz előjelet engedi meg.

A Clipper 5.0 a numerikus értékeket (a numerikus mezők kivételével) az IEEE szabvány szerinti duplapontosságú lebegőpontos formátumban tárolja,a megengedett 10^{-308} és 10^{308} közötti tartományban. A pontosság 16 értékes tizedesjegy, a numerikus értékek képernyőn történő megjelenítése 32 jegyig lehetséges (amelyből 30 a számjegy, 1 a tizedespont, és 1 az előjel helye). A 32-nél több jegyből álló számok megjelenítése explicit módon történik: a 16 értékes tizedesjegy és a 10 megfelelő hatványkitevőjének megadásával.

Amikor egy adatbázismezőből használunk értéket, az ottani numerikus forma automatikusan konvertálódik az IEEE formátumra, illetve ugyanez igaz fordítva is. Mivel a leghosszabb megjeleníthető numerikus érték a 32 jegy, ezért ez az ajánlott leghosszabb numerikus mezőhossz is.

A Clipper az alábbi formában megadott számokat tekinti numerikus értéknek:

- 1234
- 1234.5678
- -1234
- +1234.5678
- -.5678

Fontos! A karakteres változókkal ellentétben a numerikus típusú változókat nem kell határolójelek közé tenni. Ha a számjegyeket valamely határolójel közé tesszük, akkor karakteres típusként veszi figyelembe.

A numerikus műveletek

Művelet	Hatása
+	összeadás, vagy pozitív előjel
-	kivonás, vagy negatív előjel
*	szorzás
/	osztás
%	modulo
** vagy ^	hatványozás
++	növelés
—	csökkentés
= vagy ==	egyenlőség vizsgálata
!=, <> vagy #	nem egyenlő
<	összehasonlítás a kisebb érték szerint
<=	a kisebb vagy egyenlő érték
>	összehasonlítás a nagyobb érték szerint
>=	a nagyobb vagy egyenlő érték
= vagy STORE	értékadás
:=	azonnali (in-line) értékadás
+=	azonnali összeadás és értékadás
-=	azonnali kivonás és értékadás
*=	azonnali szorzás és értékadás
/=	azonnali osztás és értékadás
**= vagy ^=	azonnali hatványozás és értékadás
%=	azonnali moduloképzés és értékadás
REPLACE	a mezőérték módosítása
ABS()	abszolút érték számítása
CHR()	az ASCII érték karakteresre konvertálása
DESCEND()	komplemensre konvertálás
EMPTY()	a nulla érték ellenőrzése
EXP()	az *e* (természetes alap) hatványaira emelés
INT()	egész számra konvertálás (egész rész)
LOG()	természetes alapú logaritmus számítása
MAX()	maximum meghatározása
MIN()	minimum meghatározása

MOD()*	a dBase III Plus moduloérték számítása
ROUND()	fel- vagy lekerekítés
SQRT()	négyzetgyökvonás
STR()	karakteresre konvertálás
TRANSFORM()	formázás
TYPE()	adattípus meghatározása makróval
VALTYPE()	adattípus meghatározása direkt módon

A számok összehasonlítása aktuális numerikus értékük szerint történik, így a numerikus adattípusnál az = és az == operátor teljesen azonos. A numerikus műveletek pontosságát a SET parancsok nem befolyásolják, ezek csak a megjelenítést szabályozzák.

A tömbök

Tömbnek nevezzük az azonos nevű változók együttesét, amelynek elemeit indexeikkel érhetjük el. Az egyes elemek tetszőleges adattípusúak lehetnek — a memo típus kivételével, amely egy különleges mezőtípusnak számít. Az elemek egy tömbön belül eltérő típusúak is lehetnek, elemként tartalmazhatnak másik tömböt, vagy kódblokkot is. A TYPE() és a VALTYPE() függvények a tömbök esetében az "A" értéket adják vissza.

A tömbök egy azonosítóból és az ezt követő szögletes zárójelek közé írt egy vagy több számból állnak. A szögletes zárójel a tömbdefiníció része, kötelező megadni. A tömbnek legalább egy dimenzióval kell rendelkeznie. A többdimenziós tömbök kétféleképpen adhatók meg:

<azonosító>[<elem1>, <elem2>,...]

vagy

<azonosító>[<elem1>] [<elem2>] ...

A dimenziók mérete dinamikusan deklarálható, például:

```
LOCAL i := 0, j := 0
@ 10, 10 GET i
@ 11, 10 GET j
READ
LOCAL tomb[i, j]
```

Egy tömb létrehozható az ARRAY(), a DBSTRUCT() és a DIRECTORY() függvények segítségével is.

A tömbelemek címzése a tömb nevével és indexével történik, például a **tomb[5,30]** a tomb nevű tömb 5. sorának 30. elemét címzi. Ugyanakkor mind a név, mind az index helyén állhat olyan kifejezés, amely a megfelelő azonosítást biztosítja. Például az alábbi tömbcímzések mindegyike helyes:

```
{"a", "b", "c"}[2]
x[2]
ARRAY(3)[2]
&(<makrokifejezes>)[2]
(<komplex kifejezes>)[2]
<azonosito>[<numerikus kifejezes>]
```

Vigyázni kell a túlcímzésre, mert ha a dimenzióra deklarált értéknél nagyobb indexre hivatkozunk, az hibajelzést eredményez.

A tömbök elemei számára egyidejűleg értéket csak a deklaráláskor tudunk adni (egyidejűleg ekkor is csak a NIL értéket), illetve ehhez a megfelelő függvényeket kell alkalmazni. Egyébként az egyes tömbelemek számára az értékadó utasítások segítségével egyenként lehet értéket adni. A speciális tömbkezelő függvények közül az ARRAY() a NIL értékkel, a DBSTRUCT() az adatbázis szerkezetével, a DIRECTORY() a lemezkönyvtár információival tölti fel (a létrehozással egyidejűleg) a tömbelemeket. A két utóbbi függvény alkalmazása esetén nem kell megadni a tömb méretét, például:

```
LOCAL tomb := DIRECTORY("*.*", "D")
```

Az ilyen jellegű értékadás alól a STATIC utasítás a kivétel, amely az értékadásnál csak egyszerű kifejezéseket és konstansokat enged meg.

Az AFILL() függvény segítségével a teljes tömb, vagy annak megadott részei kaphatnak egyidejűleg értéket, például az első tíz elem az 1, a második tíz elem a 2 értéket kapja:

```
AFILL(tomb, 1, 1, 10)
AFILL(tomb, 2, 11, 10)
```

A tömbelemek értékadásánál tetszőleges adattípusú kifejezés eredménye felhasználható, beleértve a kódblokkokat és más tömböket is. Szabályos például az alábbi értékadás:

```
tomb[1][2] := {1, 2, 3, 4, 5}
```

Ha a tömböt karaktersorozatként adjuk meg, a makróhívatkozás után tömbként értelmeződik, például:

```
kartomb := "{1, 2, 3}"
ujtomb := &kartomb
? VALTYPE(ujtomb)        // Eredmény: "A"
```

Ez a megoldás akkor javasolt, ha a tömb értékeit szövegállományba kívánjuk másolni. Ekkor arra kell vigyázni, hogy a makró legfeljebb 254 karakter hosszú sorozatot tud kezelni.

A tömbök — nevükkel való hívatkozással — függvényargumentumként szerepeltethetők, illetve azok visszaadott értékei lehetnek. Például:

```
LOCAL tomb[17]              // minden elem NIL
AFILL(tomb, 0)              // minden elem 0
Novelo(tomb)                // minden elem 1 lesz
//
? tomb[5]                   // 1

FUNCTION Novelo( novtomb )
   FOR i = 1 TO LEN(novtomb)
      novtomb[i]++
   NEXT
   RETURN (novtomb)
```

Tömbök adhatók vissza eljárásból, vagy felhasználói függvényből, például:

```
tomb := Tombkesz(10, "Új érték")
//
FUNCTION Tombkesz( elemszam, ertek )
   LOCAL ujtomb[elemszam]
   AFILL(ujtomb, ertek)
   RETURN (ujtomb)
```

A tömbelemek végigjárása kétféleképpen lehetséges:

- ciklusutasítással;
- kódblokk segítségével.

A tömbelemek ciklusutasítással történő végigjárásának tipikus formája:

```
LOCAL tomb[10]
FOR i:= 1 TO 10
   tomb[i] := i
NEXT
```

A kódblokk segítségével történő végigjáráshoz az AEVAL() függvényt kell felhasználni. Az alábbi példában egy egydimenziós tömb elemeit nyomtatjuk ki:

```
AEVAL(tomb, { |elem| QOUT(elem) })
```

Többdimenziós tömb esetében egymásba ágyazott AEVAL() függvényeket kell alkalmazni:

```
LOCAL tomb[5][30], i:= 1
// "az elemek feltöltése soronként"
AEVAL(tomb, { |elem| AFILL(elem, i++) })
// "az összes elem megjelenítése"
AEVAL(tomb,{ |elem| AEVAL(elem,{ |ertek| QOUT(ertek)})})
```

Abban az esetben, ha nem tudjuk, mekkora tömbre lesz később szükségünk, üres tömb is létrehozható. Ez később az AADD() vagy az ASIZE() függvénnyel bővíthető. Példa üres tömb létrehozására:

```
LOCAL urestomb[0]
LOCAL urestomb := {}
```

Annak ellenőrzése, hogy üres tömbbel állunk-e szemben:

```
FUNCTION Urestell( valami )
   IF VALTYPE(valami) = "A" .AND. EMPTY(valami)
      RETURN (.T.)
   ENDIF
   RETURN (.F.)
```

Az üres tömb nem ellenőrízhető a NIL értékkel való összehasonlítás útján. Ez minden esetben *hamis* (.F.) értéket eredményez.

A tömbök mérete meghatározható a LEN() függvény segítségével. Többdimenziós tömb esetén a függvény az első index szerinti értéket adja vissza:

```
LOCAL tomb[10][14]
? tombmeret := LEN(tomb)        // 10
```

Ennek az a magyarázata, hogy a többdimenziós tömböket a Clipper "altömbökként" kezeli, ezért ha a következő dimenzió méretére vagyunk kíváncsiak, akkor az első elemre hivatkozással kérdezzük a hosszt:

```
? tombmeret := LEN(tomb[1])       // 12
```

Fontos! A LEN() függvény üres tömb esetén nulla értéket ad vissza.

Két elem összehasonlítása az == operátorral lehetséges. Ugyanakkor ezzel a módszerrel két tömb csak akkor bizonyul azonosnak (az összehasonlítás eredménye csak akkor lesz *igaz* (.T.)), ha a két tömb a memóriában azonos címen található, például:

```
LOCAL tomb := {1, 2, 3}
mastomb := tomb
? mastomb == tomb          // Eredmény: .T.
ujtomb := ACLONE(tomb)
? ujtomb == tomb           // Eredmény: .F.
```

Az = operátor nem alkalmazható két tömb összehasonlítására. Két tömb egyezőségének legbiztosabb megállapítási módja az elemenkénti összehasonlítás, amelyre például az alábbi függvényt alkalmazhatjuk:

```
FUNCTION Tombhas( egy, ket )
   IF LEN(egy) <> LEN(ket)      // "eltérö méret"
      RETURN (.F.)
   ENDIF
   FOR i := 1 TO LEN(egy)
      IF (egy[i] <> ket[i])     // "valamely elem eltér"
         RETURN (.F.)
      ENDIF
   NEXT
   RETURN (.T.)          // "a méret és minden elem azonos"
```

A tömbök mérete menet közben növelhető és csökkenthető. Ezt az ASIZE() függvénnyel valósíthatjuk meg:

```
LOCAL tomb[86]
ASIZE(tomb, 5)
? LEN(tomb)             // 5
ASIZE(tomb, 30)
? LEN(tomb)             // 30
```

A tömb méretének növelését eredményezi az AADD() függvény is, ekkor ugyanis egy elem adható a tomb végéhez. A függvény egyidejűleg hajtja végre a tömb méretének növelését, valamint az elem számára az értékadást:

```
LOCAL tomb[14]
AADD(tomb, 126)
? LEN(tomb)              // 15
? tomb[15]               // 126
```

Tömbelem törlése és beszúrása az ADEL() illetve az AINS() függvényekkel lehetséges, azonban ezek a függvények nem változtatják meg a tömb méretét. Az ADEL() olymódon törli a megadott indexű elemet, hogy az elemtől kezdve minden elemet egy hellyel balra tol, az utolsó elem pedig a NIL értéket veszi fel. Az AINS() pedig a megadott helyre beszúr egy NIL értékű elemet, az utolsó elem pedig elvész.

Az elemek az ACOPY() függvénnyel másolhatók, illetve az ACLONE() függvénnyel duplikálhatók. Az ACOPY() függvény a tömb elemeinek értékét másolja egy másik tömbbe, míg az ACLONE() a meglévő tömb egészéről készít másolatot. A különbséget jól mutatja az alábbi példa:

```
LOCAL egytomb[5], kettomb[17]
egytomb[1] := {1, 2, 3}
ACOPY(egytomb, kettomb)
? egytomb[1] == kettomb[1]        // .T.
```

```
hartomb := ACLONE(egytomb)
? egytomb[1] == hartomb[1]          // .F.
```

Az utóbbi hamis eredmény magyarázatát a tömbök összehasonlításával foglalkozó részben találhatjuk.

Az ASORT() függvény segítségével a tömb összes eleme, vagy annak meghatározott része nagyság szerint sorbarendezhető. Például:

```
ASORT(tomb)              // a teljes tömb rendezése
ASORT(tomb,10,5)         // a 10-14. elemek rendezése
```

Az ASORT() függvényben egy kódblokkon keresztül megadhatunk tetszőleges rendezési szempontot, például a rendezés előtt minden elemet nagybetűre alakítunk át:

```
ASORT(tomb,,, { |x, y| UPPER(x) < UPPER(y) })
```

A tömbelemek értéke szerinti keresés az ASCAN() függvénnyel lehetséges. Ennél figyelni kell arra, hogy a kis és nagybetűk között különbséget tesz. Például:

```
LOCAL tomb := { "Sue", "Lise", "Mary", "Kate" }
? ASCAN(tomb, "Lise")       // 2
? ASCAN(tomb, "lise")       // 0
```

A fenti hiba kiküszöbölésére alkalmazzuk a kódblokkot:

```
? ASCAN(tomb, { |x| UPPER(x) == "LISE" })       // 2
```

Ha egy tömbön belül azonos értékkel több elem is szerepel, akkor mindegyik megkereshető az alábbi függvény segítségével:

```
LOCAL tomb := {"Sue","Lise","Mary","Lise"}, start := 1
vege := LEN(tomb)
DO WHILE (hely := ASCAN(tomb, "Lise", start)) > 0
   ? hely, tomb[hely]
   IF (start := ++hely) > vege
      EXIT
   ENDIF
ENDDO
```

Objektumok és üzenetek

A Clipper 5.0 készítői, figyelembe véve az objektumorientált programozás térhódítását, a nyelvnek ebben a verziójában egy speciális adattípus — az objektum — fogalmát vezették be. Hangsúlyozzák azonban, hogy a Clipper 5.0 **nem objektumorientált** programozási nyelv, csupán annak néhány elemét tartalmazza!

A Clipper 5.0 objektumok egyszerű adatértékek, amelyek előre definiált szerkezettel és viselkedésmóddal rendelkeznek. A néhány előre definiált objektumtípust — amelyet osztályoknak neveznek — a rendszer kezeli. A létező osztályokon kívül újak létrehozása nem lehetséges!

Az osztályok

A Clipper 5.0 verziója az alábbi szabvány osztályokat tartalmazza:

Jelzése	Megnevezése
Error	Hibakezelő osztály
Get	Adatbeviteli osztály
TBrowse	Adatáttekintési osztály
TBColumn	Adatmegjelenítési osztály

Minden osztályhoz speciális függvények és változók tartoznak, amelyek alkalmazásával az illető osztály objektumainak állapota lekérdezhető, illetve módosítható. Ezek használata során egy-egy *üzenet*et kell továbbítani az objektum számára. Ez az üzenetküldés speciális szintaktikai szabályok betartásával, és az üzenetküldő operátorral (:) történik. Az üzenetküldés formája:

<objektum>:<szelektor> [(<argumentumlista>)]

Például:

```
sajBrowse:PageUp()
```

ahol a sajBrowse nevű változó tartalmazza az objektumleírást, a PageUp() a szelektor, amelynek hatására a konkrét példában az objektumban definiált mezőértékek megjelenítésében egy képernyőnyi előrelapozást írunk elő. A szelektor utáni zárójelek elhagyhatók, ha argumentumot nem tartalmaz, azonban célszerű kitenni a külső változóktól való megkülönböztetés érdekében. Az üzenettovábbítás után — a függvényhíváshoz hasonlóan — egy értéket kapunk vissza.

A külső változók

Minden osztályhoz speciális külső változók tartoznak. A külső változók értékei az üzenetküldő operátor speciális alkalmazásával adhatók meg, illetve használhatók fel, például:

```
row := sajBrowse:cursRow
sajBrowse:cargo := cargoErtek
```

Az objektumok alkalmazásának lehetőségeit a *„Szabvány osztályok"* fejezetben részletesebben tárgyaljuk. Példaként most két alkalmazási lehetőséget vázolunk fel:

Mint ismeretes, az adatok bekérésének egyik lehetséges módja a GET parancs alkalmazása. A GET parancs hatására a képernyő kijelölt területein különböző adatok írhatók be. Ezek az adatok mindaddig módosíthatók, amíg egy READ parancsot ki nem adunk. A READ parancsot megelőző GET-ek összessége alkotja a GET-objektumot, amely az első GET-parancs kiadásakor nyílik meg, és a READ parancs kiadásáig tart. Az objektumon belül a speciális külső változók és függvények segítségével — programunkból vezérelhetően — tetszőlegesen mozoghatunk, azok értékeit módosíthatjuk stb. akkor, amikor ezeket az értékeket a kijelölt memóriaváltozók még nem kapták meg.

Egy másik alkalmazási lehetőséget a TBrowse és a TBColumn osztályok együttesen nyújtják. A BROWSE parancs már a dBase nyelvből ismert: az aktuális adatbázis tartalma jeleníthető meg a képernyőn. A Tbrowse használatával az adatbázis összes mezője; a TBrowse és a TBColumn együttes használatával a kijelölt mezők helyezhetők el a képernyőn általunk meghatározott méretű és elhelyezkedésű ablakban. Ezek az adatok táblázatként foghatók fel, és így is

kezelhetők. Az objektumot ez a táblázat alkotja, amelyben a rendelkezésre álló változók és függvények segítségével különböző mozgásokat és műveleteket végezhetünk.

Bővebbet a „*Szabvány osztályok*" című fejezet tartalmaz.

Az adatbázis-rendszer

A Clipper adatbázis-rendszere munkaterületekre épül. Egy munkaterületen egyidőben csak egy adatbázis lehet megnyitva. Az **adatbázist** a nevükkel azonosítható *mezők*, azok *típusa*i, *hossza*i, illetve numerikus értékek esetén *tizedesjegyeik száma* alkotják.

Minden adatbázishoz, attól teljesen függetlenül **indexállományok** hozhatók létre. Ezek az adatbázis meghatározott mezőjének (vagy mezőinek) értékei szerint logikai sorrendbe fűzik az adathordozón a felvitel fizikai sorrendjében elhelyezkedő adatokat. A különböző indexállományok alkalmazása során az adatok fizikai sorrendje nem változik.

Az adatbázisok között — indexállományaikon keresztül — **relációs kapcsolatok** építhetők fel. A SET RELATION parancs alkalmazásával két vagy több adatbázis egyidejűleg kezelhető.

Az adatbázis tartalmazhat úgynevezett **memo-mezőt**, amely az adatbázishoz kapcsolódó, de attól független állományban helyezkedik el.

A munkaterületeken a fenti állományokon kívül **alternatív állomány** (lásd SET ALTERNATE parancs) is megnyitható, amely szövegállomány létrehozását teszi lehetővé. Az alternatív állomány a munkaterülettől független, egyidőben (az összes munkaterületet figyelembe véve) csak egy lehet nyitva.

A munkaterületek

A Clipper adatbázis-rendszere 255 munkaterületet tud kezelni, amelyből 250 áll a programozók rendelkezésére, 5 munkaterületet belső használatra foglal le. (Ezért kell a CONFIG.SYS-ben a FILES parancs megadásakor a szükségesnél 5-tel nagyobb számot megadni!)

Egy munkaterületen egyidőben csak egy adatbázis, az esetlegesen ehhez kapcsolódó memoállomány, maximálisan 15 indexállomány, illetve — az összes munkaterületet figyelembe véve — egy alternatív állomány nyitható meg. A munkaterület lehet *foglalt* vagy *szabad*, attól függően, hogy található-e azon nyitott adatbázis, vagy nem. **Aktuális munkaterület**nek nevezzük azt, amelyre a programban általunk kiadott „normál" (átirányító előtag nélüli) parancsok érvényesek. A program indulásakor minden munkaterület szabad, és az 1 számmal jelzett az aktuális.

Egy munkaterületet sorszámmal, vagy névvel azonosíthatunk. A kívánt munkaterület kiválasztása a SELECT paranccsal történik. A munkaterület kiválasztása után azon megnyithatunk egy adatbázist, és a szükséges egyéb (memo, index) állományokat. Vigyázzunk arra, hogy ha egy munkaterületen már volt egy nyitott adatbázis, és ugyanazon egy újat nyitunk meg, akkor az előző automatikusan lezáródik! A legnagyobb biztonságot az nyújtja (ha nem tudjuk, hogy melyik munkaterületeken nyitottunk már meg adatbázist), ha a SELECT 0 parancsot adjuk ki, vagy az adatbázist megnyitó USE parancs sorában a NEW kulcsszót alkalmazzuk.

Különböző munkaterületeken egyidőben különböző adatbázisok lehetnek nyitva. Ezek között a SELECT parancs segítségével mozoghatunk. A SELECT parancsban a munkaterület sorszámát, vagy annak helyettesítő (alias) nevét kell megadni. Az alias nevet tetszés szerint mi adhatjuk, illetve ha nem adjuk meg, akkor az adatbázis neve lesz az. Az aliasnév segítségével

lehetőség van egy másik munkaterületről elérni valamely adatbázis szükséges mezőjét. Ehhez a mezőnév elé a példa szerint az aliasnév előtagot kell illeszteni:

<aliasnév>-><mezőnév>

Az egyes munkaterületek jellemzőit, állapotát, az azokon érvényes előírásokat stb. különböző parancsokkal és függvényekkel szabályozhatjuk, illetve kérdezhetjük le. Ezek:

A munkaterület állapotával kapcsolatos parancsok, függvények

Parancs/függvény	Hatás
SET DELETED	Törölt rekordok szűrése
SET FILTER	Logikai rekordszűrés
ALIAS()	Adatbázis alias neve
BOF()	Az állománykezdet-jel helyzete
DBFILTER()	A SET FILTER értéke
DBRELATION(),DBRSELECT()	SET RELATION információ
EOF()	Az állományvég-jel helyzete
FCOUNT()	A mezők száma
FOUND()	Találatjelző
FLOCK(), RLOCK()	Zárolásjelzők
INDEXKEY()	Indexkulcs-kifejezés értéke
INDEXORD()	A SET ORDER helyzete
LOCATE, CONTINUE	Rekordmutató áthelyezés
RECCOUNT(),LASTREC()	Rekordok száma
RECNO()	Az aktuális rekord sorszáma
SELECT()	A munkaterület sorszáma

A munkaterületen megnyitott adatbázis rekordjaival különböző műveletek végezhetők. A legtöbb parancs egyszerre egy rekordon képes műveletet végezni. Amelyik rekordra a *rekordmutató* mutat, azt nevezzük **aktuális rekord**nak. Az adatbázis megnyitásakor az aktuális rekord a logikailag első rekord (amely indexállomány nyitása esetén az indexkulcs szerinti első rekord, annak hiányában a fizikailag elsőként felvitt rekord).

A parancsok egy része egyszerre több rekordon képes műveletet végrehajtani. Ezeknek a kijelölt rekordoknak a körét a *hatókör* megadásával szabályozhatjuk:

Hatókör	Értelmezés
ALL	Az összes rekord
NEXT <szám>	Az aktuálissal együtt <szám> db rekord
RECORD <szám>	A megadott sorszámú rekord
REST	Az aktuálistól kezdve az összes rekord
FOR <feltétel>	A feltételt teljesítő rekordok
WHILE <feltétel>	Az összes rekord amíg igaz a feltétel

A különböző parancsoknál különböző a hatókör alapértelmezése.

Az indexállományok

Az indexállományok segítségével állíthatók — fizikai áthelyezés nélkül — a szükséges sorrendbe az adatbázis rekordjai. Egyidejűleg maximum 15 indexállomány lehet egy adatbázishoz nyitva.

Az indexállomány szerinti sorrendet az **indexkulcs** határozza meg. Az indexkulcs maximum 250 karakter hosszú, karakteres, dátum vagy numerikus típusú lehet. Ha az indexkulcsban különböző típusú mezőket kívánunk egyidejűleg megadni, akkor azokat a megfelelő függvények segítségével azonos típusúra kell konvertálni.

A UNIQUE paraméterrel, illetve a SET UNIQUE ON kapcsoló állításával lehetőségünk van arra, hogy az azonos kulcsú rekordok közül csak egy — az első — legyen az indexállományban figyelembe véve.

Az egyidejűleg megnyitott indexállományok közül **ellenőrző indexnek** nevezzük azt, amely szerint a rekordokat logikailag sorbarendezettnek látjuk. A különböző indexállományok között a SET ORDER parancs segítségével válthatunk. A SET ORDER TO 0 — az indexállományok nyitvahagyása mellett — a fizikai sorrendet veszi figyelembe. Az adatbázis mezőin végzett módosítások (akkor is, ha az valamely indexállomány szerinti indexkulcsot érinti) egyidejűleg minden nyitott indexállományban végrehajtásra kerülnek.

Az indexállományok segítségével az adatbázisban igen gyors keresés érhető el. Erre a célra a SEEK és a FIND parancs szolgál. Ha létezik az adatbázisban a keresett állomány, a FOUND() függvény értéke *igaz* (.T.) lesz, és a rekordmutató a keresett rekordra mutat. Ha nincs ilyen rekord, akkor a rekordmutató az állomány végére megy: az EOF() értéke *igaz* (.T.) lesz. Ha a keresett rekord nem található, a SET SOFTSEEK kapcsoló beállításával lehetőségünk van arra, hogy a keresettnél nagyobb, ahhoz legközelebb álló rekordra lépjünk. A FOUND() értéke ekkor *hamis* (.F.), de az EOF() értéke csak akkor lesz *igaz* (.T.), ha nincs a keresettnél nagyobb kulcsú rekord.

Nemindexelt állományon, vagy az indexkulcstól eltérő mező szerint csak a LOCATE paranccsal kereshetünk!

Az input/output rendszer

A Clipper programozási nyelv teljes rendszert tartalmaz az információk bevitelére, és képernyőn, nyomtatón vagy lemezállományban történő megjelenítésére . Mind az adatbevitel (input), mind a megjelenítés (output) *soros* és *direkt* módon történhet. *Soros* mód esetén a megjelenítés, illetve adatbevitel az aktuális képernyőprompttól, *direkt* mód esetén a megadott sor- és oszlopkoordinátától történik. Soros mód esetén lehetőség van az információ egyidejű megjelenítésére képernyőn, nyomtatón és lemezállományban, míg direkt mód esetén csak a képernyőn, vagy csak a nyomtatón (illetve a nyomtatót szimulálva lemezállományban) jeleníthető meg. A soros mód szerint működő parancsokat, függvényeket *konzolműveleteknek* nevezzük:

Konzolműveletek

Művelet	Hatás
?|??	Egy vagy több kifejezés eredménye
ACCEPT	Karakteres input várása
DISPLAY|LIST	Az adatbázis tartalma
INPUT	Tetszőleges adattípus várása
LABEL FORM	Az adatbázis címke szerinti tartalma
REPORT FORM	Jelentéskészítés az adatbázisról
TEXT...ENDTEXT	Szövegblok
TYPE	Egy állomány tartalma
WAIT	Egy karakterre várás
QOUT()|QQOUT()	Egy vagy több kifejezés eredménye

A képernyőn történő megjelenítés a SET CONSOLE OFF paranccsal letiltható.

A teljes képernyős (*direkt*) műveletek az előzőektől abban térnek el, hogy a képernyő (vagy nyomtató) megadott sorától és oszlopától kezdődik a megjelenítés.

Művelet	Hatás
@...BOX	Keretrajzolás
@...CLEAR	Képernyő(rész) törlése
@...GET	Adatbekérés
@...SAY	Kifejezés eredményének megjelenítése
@...TO	Keretrajzolás
CLEAR	Képernyőtörlés
COL()	Az aktuális oszloppozíció
ROW()	Az aktuális sorpozíció
MAXCOL()	A maximális oszloppozíció
MAXROW()	A maximális sorpozíció
SAVESCREEN()	Képernyőterület mentése
RESTSCREEN()	Képernyőterület visszatöltése

A képernyő, illetve annak egy részének színe programból állítható. Az aktuális szín elmentése, illetve későbbi visszatöltése a SETCOLOR() függvény segítségével lehetséges. A @...GET parancs kivételével minden input/output művelet a szabvány színt használja.

A *soros* módon történő input/output műveletek a SET PRINTER ON kapcsolóállítással (illetve némely műveletnél a TO PRINTER paraméterrel) nyomtatóra irányíthatók. A nyomtatóra irányítás esetén a PROW() és a PCOL() függvények mindig az aktuális nyomtatófej pozíciót (sort és oszlopot) adják vissza. Ugyanezeket az értékeket a képernyőre vonatkozóan a ROW() és a COL() függvények tartalmazzák. A nyomtatási margó a SET MARGIN paranccsal állítható.

Lemezállományba elvileg csak a soros mód szerinti output irányítható, de lehetőség van arra, hogy a nyomtatót egy lemezállománnyal helyettesítsük. Ennek módját a következő példa világítja meg:

```
SET PRINTER TO Suenyom.txt
SET DEVICE TO PRINTER
//
// <nyomtatás>
//
SET DEVICE TO SCREEN
SET PRINTER TO
```

A soros módon megvalósított output lemezállományba írásához (ha a művelet nem rendelkezik TO FILE paraméterrel) egy alternatív állományt kell megnyitni:

```
SET ALTERNATE TO Suenyom.txt
SET ALTERNATE ON
//
// <nyomtatások>
//
SET ALTERNATE OFF
CLOSE ALTERNATE | SET ALTERNATE TO
```

A billentyűzetrendszer

A Clipper egy pufferben automatikusan tárolja a billentyűzeten lenyomott karaktereket, amelyeket később a lenyomás sorrendjében használ fel. A billentyűzeten lenyomott karakterek felhasználása egy várakozási állapotot előidéző parancsnál vagy függvénynél, illetve az INKEY() függvénynél következik be. Várakozási állapotot az ACCEPT, INPUT, READ, WAIT, ACHOICE(), DBEDIT() és MEMOEDIT() idéz elő.

A billentyűzetpuffer mérete a SET TYPEHEAD paranccsal 0 és 32767 között változtatható. A 0 értékkel gyakorlatilag lezárjuk a billentyűzetpuffert, és adatbevitel csak akkor lehetséges (ekkor is direktben), ha egy művelet azt kéri. A SET TYPEHEAD parancs a pufferméret módosítása előtt törli annak tartalmát.

A KEYBOARD paranccsal programból helyezhetünk el karaktereket a billentyűzetpufferben. Kiadásakor először törli a puffer tartalmát. Különösen jól alkalmazható demonstrációs programoknál az ACHOICE() függvénnyel együtt.

A billentyűzetpuffer tartalmáról az INKEY(), LASTKEY() és a NEXTKEY() függvények adnak információt. Ezek a függvények -39 és 386 közötti numerikus értéket adnak vissza, amely az illető karakter INKEY() kódját (lásd függelék) tartalmazza. Ezzel a kóddal egyértelműen azonosíthatók az Alt-, Ctrl-, és Shift- kombinációk is.

Az **INKEY()** várakozási állapotot idéz elő, és a következőnek beütött billentyű(kombináció) ASCII kódját adja vissza, egyidejűleg azt törli a billentyűzetpufferből.

A **NEXTKEY()** abban tér el az INKEY()-től, hogy nem törli a kódot a billentyűzetpufferből.

A **LASTKEY()** a billentyűzetpufferből legutóbb eltávolított kódot mutatja meg.

Előre definiált billentyűk

A SET KEY parancs segítségével lehetőségünk van arra, hogy tetszőleges billentyűhöz egy eljárás végrehajtását rendeljük. Az illető billentyű lenyomása esetén egy várakozási állapot elérésekor az általunk definiált eljárás végrehajtása kezdődik meg. A SET KEY — paraméterek

nélküli — kiadásával az összes eddigi átdefiniálás érvényét veszti. Az eljáráshoz három paraméter automatikusan kapcsolódik (PROCNAME(), PROCLINE(), READVAR()).

A SET FUNCTION paranccsal — a SET KEY-hez hasonlóan — a funkcióbillentyűk lenyomásához rendelhetünk egy-egy eljárásindítást. Ha a SET FUNCTION és a SET KEY azonos billentyűt definiál át, akkor a SET FUNCTION szerinti marad érvényben (ez a korábbi verziókhoz képest eltérés).

Egyidejűleg 32 billentyű definiálható át. Alapértelmezésben az F1 funkcióbillentyűhöz a HELP van rendelve.

Alacsonyszintű állománykezelés

A Clipper több függvényével biztosítja a DOS bináris állományok programból történő direkt kezelését. Lehetőségei:

- állomány megnyitása — FOPEN(), FCREATE();
- új állomány létrehozása — FCREATE();
- olvasás nyitott állományból — FREAD(), FREADSTR();
- írás nyitott állományba — FWRITE();
- a file-pointer mozgatása — FSEEK();
- állomány zárása — FCLOSE().

Az alacsonyszintű állománykezelésnél nagyon fontos arról meggyőződnünk, hogy a végrehajtott művelet sikeresen fejeződött-e be. Ezért a függvényhívás után ellenőrizzük azok visszaadott értékeit, vagy az FERROR() függvénnyel vizsgáljuk meg a DOS hibaüzenetszámot (helyes végrehajtás esetén ez 0). Az alábbi táblázat mutatja azokat az értékeket, amelyeket az egyes függvények hiba esetén adnak vissza:

Függvény	Hibajelzés értéke
FCLOSE()	.F.
FCREATE()	-1
FOPEN()	-1
FREAD()	0, vagy a beolvasandónál kevesebb byte
FREADSTR()	""
FSEEK()	Nincs hibajelzése
FWRITE()	0, vagy a kiírandónál kevesebb byte.

A BROWSE és a GET rendszerek

A Clipper 5.0 új BROWSE és GET rendszerének sajátossága az objektum típusú adatkezelés. Osztályai: a TBrowse és a TBColumn, illetve a GET. Ezekről részletesebben a *„Szabvány osztályok"* fejezetben esik szó.

Clipper utasítások

BEGIN SEQUENCE

Utasításblokk (szekvencia) létrehozása

BEGIN SEQUENCE
<utasítások>...
[BREAK [<kifejezés>]]
<utasítások>...
[RECOVER [USING <változó>]]
<utasítások>...
END [SEQUENCE]

BREAK <kifejezés> — kilépés az utasításblokkból, és a RECOVER (ha van), vagy az END után következő utasítást hajtja végre, miközben a program számára visszaadott érték a <kifejezés>-nek megfelelő lesz.

RECOVER USING <változó> — Egy folytatási pont a BREAK után. Ha a USING <változó> paramétert megadtuk, akkor a <változó> értéke a BREAK utáni <kifejezés> értékét veszi fel.

END — az utasításblokk vége. A program a szekvencia utáni utasításon folytatódik.

A BEGIN SEQUENCE ... END a szorosan összetartozó utasításokat egy csoportba (szekvenciába) fogja össze. Ha a program végrehajtása során egy BREAK-hez érkezik, akkor a vezérlés vagy a legközelebbi RECOVER utasításra (ha van ilyen), vagy az END után következő utasításra adódik át. Leggyakoribb alkalmazási módja hibakezelési eljárásoknál, ahol például egy hiba előfordulása esetén a BREAK utáni <kifejezés> tartalmazza a hibakódot, a RECOVER utáni utasítások pedig ennek lekezelését végzik. Ha a program a végrehajtás során úgy találkozik egy RECOVER utasítássorozattal, hogy nem BREAK után lépett rá, akkor ezek figyelmen kivül maradnak, és az END utáni első utasítás következik.

A szekvenciából egy RETURN paranccsal ki tudunk lépni, de a LOOP és EXIT utasításokkal nem. Ha a BEGIN SEQUENCE egy DO WHILE ... ENDDO, vagy FOR ... NEXT cikluson belül található, akkor a LOOP vagy EXIT egy RECOVER után megadva a szekvenciára vonatkozik, és annak elejére vagy végére megy (lásd példa).

A szekvenciákat rugalmasan használhatjuk: egymásba ágyazhatók, illetve tetszőleges számban alakíthatók ki egy programon, eljáráson vagy felhasználó által készített függvényen belül. Több szekvencia esetén mindegyik egyetlen műveletként fogható fel.

Példák:

Egy szokványos szekvencia, ahol a BREAK utáni feltétel teljesülése esetén a vezérlés a RECOVER utáni utasításra tevődik át

```
BEGIN SEQUENCE
   <utasítások>...
   IF felt
      BREAK
   ENDIF
```

```
RECOVER
   <folytató utasítások>...
END
<folytató utasítások>...
```

Hiba lekezelésének bemutatása a RECOVER USING-on keresztül történő paraméterátadással, ami a hiba keletkezésének helyét tartalmazza:

```
LOCAL hibhely, uthandl
//
// "Az elözö handler mentése, az új beállítása"
uthandl := ERRORBLOCK({ |hiba| Sajhandl(hiba, .T.) })
//
BEGIN SEQUENCE
   .
   . <müveletek>
   .
RECOVER USING hibhely
   //
   // "Üzenet küldése a hibahelynek, és a hiba lekezelése"
   ? "Hiba: "
   IF hibhely:kodok != 0
      ?? hibhely:hibaleiras
   ENDIF
   .
   .
   .
END
//
// "Az elözö handler visszatöltése"
ERRORBLOCK(uthandl)

FUNCTION Sajhandl( hibakod, helyihandl)
   //
   // "A hiba jelzése"
   IF helyihandl
      BREAK hibakod
   ENDIF
   .
   . <egyéb hibakezelö utasítások>
   .
RETURN NIL
```

Példa a LOOP utáni vezérlésátadásra:

```
DO WHILE .T.
   BEGIN SEQUENCE
      .
      . <utasítások>
      .
   RECOVER
```

```
      IF Folyt()
         LOOP        // "A szekvencia ismétlése"
      ENDIF
   END
   EXIT              // "Kilépés a ciklusból"
ENDDO
```

Lásd még: RETURN, ERRORBLOCK()

DECLARE*

Private memóriaváltozók és tömbök definiálása és kezdeti értékadása

DECLARE <azonosító> [[:= <érték>], ...]

<azonosító> — a *private* változó vagy a *tömb* neve. Ha az <azonosító> után szögletes zárójel ([ ]) következik, akkor tömb keletkezik, amelynek méretét a szögletes zárójelek közé írt szám(ok) jelzi(k), például tomb[<elem1>,<elem2>,...], vagy tomb[<elem1>][<elem2>]... Az elemek maximális száma dimenziónként 4096.

<érték> — a deklarált változó vagy tömb kezdeti értéke, amelyet nem kötelező megadni. Ha megadjuk, akkor az értékadó operátor (:=) után kell állnia. Tetszőleges Clipper kifejezés lehet, beleértve a karaktersorozatot is. Ha nincs megadva, akkor a változó a típusától függő NIL értéket, tömb esetében pedig minden elem a megfelelő NIL értéket veszi fel. Tömbnek nem lehet a deklarálás során nevének megadásával értéket adni.

Egy DECLARE utasítással több változó és tömb is definiálható, ezeket vesszővel kell egymástól elválasztani.

A DECLARE a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem javasolt! Használjuk helyette a PRIVATE utasítást.

Példák:

```
DECLARE tomb1[17], tomb2[14, 17], tomb3[14][17][20]
DECLARE valt1, valt2 := SPACE(10)
```

Lásd még: PRIVATE

DO*

Eljárás meghívása

DO <eljárásnév> [WITH <argumentumlista>]

<eljárásnév> — az indítani kívánt eljárás neve.

WITH <argumentumlista> — az eljárásnak átadni kívánt paraméterértékek. Legfeljebb 128, egymástól vesszővel elválasztott, argumentumot tartalmazhat, amelyek tetszés szerint lehetnek egyszerű változók, mezőnevek, tömbök, tömbelemek és kifejezések.

A DO utasítás meghív egy eljárást, és opcionálisan átadja annak az argumentumlistában megadott értékeket. Működése megegyezik a felhasználó által definiált függvények meghívási

módjával, azzal a különbséggel, hogy a függvényhívásnál az alapértelmezés szerinti értékeket nem kell átadni. A mezőváltozók neveit zárójelbe kell tenni, vagy a FIELD-> előtéttel kell azonosítani.

Ha az argumentumok száma nem egyezik meg a hívott eljárás paramétereinek számával, akkor:

- kevesebb argumentum megadása esetén a további paraméterek értéke NIL;
- több argumentumérték megadása esetén a felesleges értékek figyelmen kívül maradnak.

Célszerű a paraméterek számát a PCOUNT() függvénnyel előzetesen ellenőrízni.

Az argumentumok megadása kétféleképpen lehetséges:

- érték szerinti paraméterátadás, amikor a hívott rutin az értéket tetszés szerint módosíthatja;
- az értéket tartalmazó változó címének átadása, amely a változónév elé tett @ jellel történik. Ekkor ha a rutin bármit változtat az értéken, az a változóban is módosul.

A fordítás során a DO utasítás után megadott eljárás a hívás helyén befordításra kerül. Kivételt képez ez alól, ha a fordítást a /M paraméterrel végezzük, vagy a megadott nevű program fordítása már megtörtént. Ha az eljárásnévvel megegyező .prg kiterjesztésű állomány nem található az aktuális könyvtárban, akkor ezt a hívatkozást a fordító EXTERNAL-ként kezeli, és a megadott nevű tárgymodult (.obj kiterjesztésű lefordított állomány, vagy .lib könyvtár része) a szerkesztés során kell a programrendszerhez illeszteni. A DO utasítással meghívott eljárás Clipper, C, vagy Assembler nyelven írt rutin egyaránt lehet, a két utóbbi esetben azonban csak az EXTERNAL összeszerkesztés lehetséges!

A DO a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem javasolt! Helyette inkább a függvényhívási formát használjuk!

Példák:

Paraméter nélküli eljáráshívás:

```
DO Sueprog
Sueprog()      // a javasolt forma
```

Eljáráshívás két konstansérték átadásával:

```
DO Sueprog WITH "860530", "551014"
Sueprog("860530", "551014")   // a javasolt forma
```

A paraméterátadás két formája:

```
szam = 17
DO Sueprog WITH szam + 7, szam
Sueprog(szam + 7, @szam)      // a javasolt forma
```

Ha bizonyos paramétereknek nem kívánunk értéket átadni:

```
DO Sueprog WITH ,,,,"910219"
Sueprog(,,,,"910219")      // a javasolt forma
```

Lásd még: FUNCTION, LOCAL, PARAMETERS, PRIVATE, PROCEDURE, PUBLIC, RETURN

DO CASE

Többszörös elágaztatási lehetőség

DO CASE
CASE <feltétel1>
** <utasítások> ...**
[CASE <feltétel2>]
** <utasítások> ...**
[OTHERWISE]
** <utasítások> ...**
END[CASE]

CASE <feltételx> — egy utasításblokkot definiál, amely akkor kerül végrehajtásra, ha a <feltétel> értéke igaz (.T.)

OTHERWISE — egy olyan utasításblokk definiálása, amely akkor kerül végrehajtásra, ha a CASE ágakon megadott feltételek egyike sem igaz.

A DO CASE ... ENDCASE szerkezettel többszörös elágazási lehetőséget vizsgálhatunk meg, és a teljesülő feltételhez kapcsolódó utasításokat hajthatjuk végre. A program sorban ellenőrzi a CASE utáni feltételeket, és az első igaz (.T) feltétel utáni utasításokat hajtja végre (akkor is, ha esetleg több CASE-ágon is teljesülne a feltétel), majd a vezérlést átadja az ENDCASE utáni első utasításnak. A kiválasztott CASE ágon az utasítások végrehajtása addig folytatódik, amíg a program másik CASE, OTHERWISE vagy ENDCASE kulcsszót nem talál.

Ha a CASE ágakon megadott feltételek egyike sem teljesül, akkor az OTHERWISE ágon megadott utasításokat hajtja végre. Ha nem adunk meg OTHERWISE ágat, akkor az ENDCASE utáni első utasítás végrehajtása következik.

A CASE ágak száma gyakorlatilag nincs korlátozva. Az egyes ágak tartalmazhatnak beágyazott DO CASE utasításokat, valamint más ciklusokat (DO WHILE, FOR) is. Ezek alkalmazásánál azonban figyeljünk a beágyazás szabályaira!

A DO CASE ... ENDCASE utasítás azonos az IF ... ELSEIF ... ENDIF utasítással.

Példa:

Egy gyakori alkalmazás: a kiválasztott menüpont szerinti felhasználói rutin hívása

```
@ 3, 25 PROMPT "1. rutin"
@ 5, 25 PROMPT "2. rutin"
MENU TO valasz
DO CASE
CASE valasz = 0
   RETURN
CASE valasz = 1
   Rutin1()
CASE valasz = 2
   Rutin2()
ENDCASE
```

Lásd még: BEGIN SEQUENCE, DO WHILE, FOR, IF, IF()

DO WHILE

Utasítások ciklikus ismétlése megadott feltétel teljesülése esetén

DO WHILE <feltétel>
** <utasítások> ...**
** [EXIT]**
** <utasítások> ...**
** [LOOP]**
** <utasítások> ...**
END[DO]

<feltétel> — logikai kifejezés. A DO WHILE ... ENDDO közötti utasítások mindaddig ciklikusan ismétlődve végrehajtásra kerülnek, amíg a feltétel igaz.

EXIT — feltétel nélküli kiugrást eredményez a DO WHILE vagy a FOR ... NEXT ciklusból. A vezérlés a legközelebbi ENDDO (vagy NEXT) utáni első utasításra tevődik át.

LOOP — hatására a program a legutóbb kiadott DO WHILE (vagy FOR) utasításra ugrik visszza, kihagyva ezáltal ebben a ciklusban a LOOP és az ENDDO (vagy NEXT) közötti utasításokat.

A DO WHILE ... ENDDO egy feltétel teljesüléséhez kötött ciklus. A köztük levő utasítások mindaddig ismétlődnek, amíg a feltétel igaz. Kivételt képez ez alól:

- ha a program egy EXIT utasításra lép, ekkor a vezérlés a feltétel teljesülésétől függetlenül az ENDDO utáni első utasításnak adódik át;
- ha a program egy LOOP utasításra lép, ekkor a vezérlés a ciklus kezdetének (a DO WHILE utasításnak) adódik át.

Ha a <feltétel> *hamis* (.F.), akkor a vezérlés az ENDDO utáni első utasításra adódik át, ez érvényes arra az esetre is, ha már az első feltételellenőrzés hamis értéket ad.

A DO WHILE ciklusba tetszés szerint ágyazhatunk be újabb DO WHILE vagy egyéb ciklusokat, ezeknél azonban figyelemmel kell lenni a beágyazás szabályaira!

Példák:

A DO WHILE ciklusok egymásba ágyazásának tipikus esete: kilistázunk egy állományt, és vállalatonként összegzünk. A listázás befejezéséhez az állomány-vég (EOF) jelzést figyeljük:

```
LOCAL regiceg, osszeg
USE Forgalom INDEX Cegforg NEW
DO WHILE .NOT. EOF()
   regiceg = Forgalom->ceg
   osszeg = 0
   DO WHILE regiceg = Forgalom->ceg .AND. (.NOT. EOF())
      ? Forgalom->ceg, Forgalom->ertek
      osszeg = osszeg + Forgalom->ertek
      SKIP
   ENDDO
   ? regiceg, "összesen: ", osszeg
ENDDO
```

```
CLOSE Forgalom
```

A következő példa a LOOP alkalmazását szemlélteti: ha az IF utáni feltétel teljesül, akkor a további utasításokat abban a ciklusban nem kell végrehajtani

```
DO WHILE <feltétel>
   <utasítások>
   IF <közbensö feltétel>
      LOOP
   ENDIF
   <további utasítások>
ENDDO
```

A DO WHILE szerinti feltétel a ciklus belsejében az alábbi módon is állítható:

```
LOCAL felt := .T.
DO WHILE felt
   <utasítások>...
   felt = (<feltétel>)
ENDDO
```

A leggyakoribb alkalmazási forma, amikor egy adatbázis teljes állományával kell ugyanazokat az utasításokat végrehajtani:

```
DO WHILE .NOT. EOF()
   <utasítások>
   SKIP
ENDDO
```

Lásd még: BEGIN SEQUENCE, DO CASE, FOR, IF, RETURN, DBEVAL(), IF()

EXTERNAL

A szerkesztés során illesztett külső eljárások, vagy felhasználói függvények felsorolása

EXTERNAL <eljáráslista>

<eljáráslista> — azoknak az eljárásoknak, felhasználói függvényeknek, illetve formátumeljárásoknak a felsorolása, amelyeket az .EXE állomány létrehozásához a szerkesztő programnak figyelembe kell vennie.

Az EXTERNAL utasítás a szerkesztő program (linker) számára ad át információkat, ezért kötelezően a program, vagy függvény első végrehajtható utasításaként kell szerepelnie!

Kötelező megadni az alábbi esetekben:

- ha az eljárás, felhasználói függvény, vagy formátumeljárás neve a programban csak makrókifejezésben, vagy változóértékként szerepel;
- a REPORT és a LABEL FORM parancsokon belül megadott eljárások, illetve felhasználói függvények esetében, ha azok forráskódját nem maga a hívó program tartalmazza;
- ha a felhasználói függvényt indexkulcsban alkalmazzuk, és forráskódját nem maga a program tartalmazza;

- az ACHOICE(), DBEDIT() vagy MEMOEDIT() függvényeknél alkalmazott felhasználói függvények esetén.

Az EXTERNAL deklarációk csoportosíthatók egy-egy „header" állományban, amelyek az #include utasítással egyszerre aktivizálhatók.

Példa:

Egy tipikus „header" állomány, amely az EXTERNAL parancs(ok) helyett egy utasítással hívható be:

```
// Az Externals.ch tartalma:
EXTERNAL HARDCR
EXTERNAL TONE
EXTERNAL MEMOTRAN
EXTERNAL STRTRAN
```

Ezek helyettesíthetők a következővel:

```
#include Externals.ch
```

Lásd még: #include

FIELD

Adatbázis mezőneveinek deklarálása

FIELD <mezőnévlista> [IN <aliasnév>]

<mezőnévlista> — a fordítóprogram számára mezőnévként deklarált azonosítónevek.

IN <aliasnév> — annak meghatározása, hogy a mezőnevek melyik adatbázishoz tartoznak. Ha nem adjuk meg, akkor az aktuális adatbázisra vonatkoznak, ugyanúgy, mintha az azonosító elé a FIELD-> előtag lenne illesztve.

A FIELD utasítás segítségével érhetjük el, hogy megadott azonosítókat a fordítóprogram eleve mezőnévként kezel. Az ilymódon deklarált azonosítók elé a használat során nem szükséges kitenni az aliasnevet, illetve a FIELD-> előtagot. Mint minden deklaráló utasítás, a FIELD sincs hatással a makrókban és a változóértékként szereplő azonosítókra.

A FIELD nem ellenőrzi, hogy a megadott mezőnevek valóban szerepelnek-e az adatbázisban. Ha ilyen nemlétező mezőnévre valamely későbbi utasítás során hívatkozás történik, akkor a hibajelzés futás közben ennél az utasításnál jelentkezik.

A FIELD utasítás hatóköre arra az eljárásra vagy függvényre terjed ki, amelyben kiadtuk, illetve arra a programra, amely ezt a PROCEDURE-t, vagy FUNCTION-t tartalmazza, és a fordítóprogram indításánál megadtuk a /N paramétert.

A FIELD utasítás tetszőleges végrehajtható utasítást, eljárást, vagy függvényt tartalmazhat, ha annak visszaadott értéke egy mezőnév.

A FIELD utasítást használjuk együtt a fordítóprogram /W paraméterével.

Példa:

A FIELD utasítás különböző megadásával mind az aktuális, mind a definiált adatbázis mezőnevei deklarálhatók:

```
FUNCTION mezokiir
   FIELD sorsz, cegnev, ugynok
   FIELD datum, valtkod IN Forg
   USE Forg NEW
   USE Torzs NEW
   //
   ? cegnev     // Torzs->cegnev
   ? datum      // Forg->datum
   //
   CLOSE Torzs
   CLOSE Forg
RETURN NIL
```

Lásd még: FUNCTION, LOCAL, MEMVAR, PROCEDURE, STATIC

FOR

Ciklus végrehajtása meghatározott számú alkalommal

FOR <számláló> = <kezdőérték> TO <végérték> [STEP <lépésköz>]
<utasítások> ...
[EXIT]
<utasítások> ...
[LOOP]
NEXT

<számláló> — annak a változónak a neve, amelynek értéke a ciklus következő végrehajtása előtt ellenőrzésre kerül. Ha korábban nem lett deklarálva, vagy a hívatkozás helyén „nem látható", akkor *private* változóként definiálódik.

<kezdőérték> — a <számláló> kezdeti értéke. Ha a <lépésköz> pozitív, akkor kisebbnek, ha negatív, akkor nagyobbnak kell lennie a <végérték>-nél.

TO <végérték> — a <számláló> utolsó olyan értéke, amellyel a ciklus még végrehajtásra kerül.

STEP <lépésköz> — az az érték, amellyel a <számláló> értéke a ciklus minden egyes lefutása után növelődik, mindaddig, amíg túl nem lépi a <végérték>-et. Egyaránt lehet pozitív, vagy negatív érték. Alapértelmezése: 1.

EXIT — feltétel nélküli kilépést biztosít a ciklusból, függetlenül a <számláló> aktuális értékétől. Hatására a program a legközelebbi NEXT vagy ENDDO utáni következő utasítással folytatódik.

LOOP — hatására a vezérlés a legutóbb kiadott FOR vagy DO WHILE utasításra adódik vissza.

A FOR ... NEXT utasítás segítségével a két kulcsszó közé zárt utasításokat meghatározott számú alkalommal ismételhetjük meg. A ciklus minden egyes végrehajtása során ellenőrzésre ke-

rül, hogy a <számláló> értéke nem lépte-e át a <végérték>-et (pozitív vagy negatív lépésköztől függően annál nagyobb, vagy kisebb-e). Ha igen akkor a program végrehajtása a NEXT utáni első utasítással folytatódik. Ugyanez következik be — a <számláló> értékétől függetlenül — ha egy EXIT utasítással találkozik.

A ciklusban résztvevő minden érték (számláló, kezdőérték, végérték, lépésköz) lehet egy szám, vagy numerikus kifejezés.

A FOR ... NEXT ciklusok tetszőleges számban egymásba ágyazhatók, illetve tetszőlegesen tartalmazhatnak egyéb ciklikus utasításokat is, ezeknél azonban figyelemmel kell lenni a beágyazás szabályaira.

Példák:

A FOR ... NEXT ciklus gyakori alkalmazása tömbök elemei számára történő értékadás. Az első példa a tömbelemek növekvő sorszám szerinti kezelését mutatja:

```
tombmeret = LEN(tomb)
FOR i = 1 TO tombmeret
   <utasítások> ...
NEXT
```

Az előző példa csökkenő sorrendben történő kezelésre:

```
tombmeret = LEN(tomb)
FOR i = tombmeret TO 1 STEP -1
   <utasítások> ...
NEXT
```

Lásd még: BEGIN SEQUENCE, DO CASE, DO WHILE, IF, AEVAL()

FUNCTION

Felhasználói függvény leírása

[STATIC] FUNCTION <függvénynév>[(<paraméterlista>)]
[LOCAL <azonosító> [[:= <kezdőérték>], ...]]
[STATIC <azonosító> [[:= <kezdőérték>], ...]]
[FIELD <azonosító lista> [IN <aliasnév>]]
[MEMVAR <azonosító lista>]
.
. <végrehajtandó utasítások>
.
RETURN <kifejezés>

<függvénynév> — felhasználói függvény neve. Tetszőleges hosszú lehet, de csak az első tíz karaktert veszi figyelembe; betűket, számokat és az aláhúzásjelet tartalmazhatja.

<paraméterlista> — egy vagy több paraméterváltozó felsorolása. Az itt felsorolt változók *local* deklarációt kapnak.

STATIC FUNCTION — egy olyan felhasználói függvényt deklarál, amely **csak** abból a program (.prg) állományból érhető el, amelyik azt tartalmazza.

LOCAL — Az aktuális függvényben érvényes változók és tömbök deklarálása és igény szerint kezdőértékkel ellátása.

STATIC — Azoknak a változóknak és tömböknek a deklarálása (és igény szerint kezdőértékkel ellátása), amelyek az adott függvényen kívül a program más részein is érvényesek.

FIELD — a mezőnévként használt azonosítók listája. Az IN kulcsszó alkalmazásával megadható annak az adatbázisnak a neve (alias neve), amely a deklarált mezőt tartalmazza. Egyébként mindig a hívatkozás helyén aktuális munkaterületen megnyitott adatbázis mezőneveként értelmezi.

MEMVAR — a *private* vagy *public* memóriaváltozók vagy tömbök deklarálása.

RETURN <kifejezés> — a felhasználói függvényből való kilépést jelzi, ahol <kifejezés> a hívott eljárás vagy felhasználói függvény által a hívó programnak (eljárásnak, függvénynek) visszaadott érték. Minden függvény végén kötelezően kell szerepelnie egy RETURN-nek, de a rutinon belül tetszőleges helyen több RETURN is előfordulhat.

A FUNCTION utasítással egy felhasználói függvény leírását vezetjük be. A hívó eljárásból a paraméterlistán keresztül tudunk értékeket átadni. A függvény által visszaadott értéket (a függvény eredményét) a RETURN utáni <kifejezés> tartalmazza. A Clipperben a FUNCTION és a PROCEDURE lényegében azonos feladatokat lát el, azzal a különbséggel, hogy a FUNCTION esetében kötelező értéket visszaadni a hívó eljárásnak. Ez a visszaadott érték tetszőleges adattípus lehet, beleértve a tömböt, a kódblokkot vagy a NIL értéket. Egy felhasználói függvény végét jelzi ezen kívül egy másik FUNCTION, egy PROCEDURE utasítás, vagy az állomány-vég jel. Ez alapján felhasználói függvények *leírása* nem ágyazható egymásba, de egy felhasználói függvényből meghívhatunk másikat, sőt rekurzív módon önmagát is.

A függvények elérhetőség szempontjából két csoportra oszthatók: a programrendszer tetszőleges helyéről hívható függvények a FUNCTION kulcsszóval kezdődnek, míg azok a függvények, amelyek csak az azt tartalmazó programból érhetők el — a STATIC FUNCTION kulcsszavakkal kezdődnek.

A felhasználói függvények hívása

A függvény hívása annak nevével és az argumentumlistával történik:

- <függvénynév>([<argumentumlista>])

vagy

- <aliasnév>->(<függvénynév>(<argumentumlista>))

ez utóbbi esetben a függvénynevet és az argumentumlista a megadott módon zárójelbe kell tenni.

A paraméterek

A felhasználói függvény paramétereket tud átvenni a hívó eljárásból, felhasználói függvényből, vagy a DOS-tól. Kétféleképpen adható meg: vagy a paraméterlistában, vagy a PARAMETERS utasítás után, de a kétféle megadási mód nem keverhető — ez a fordítás során lekezelhetetlen hibát eredményez!

A függvény a paraméterértékeket az argumentumlistában megadott sorrendben veszi figyelembe. A megadott értékek számának nem kell megegyezni: ha több argumentumérték lett megadva, mint ahány paraméter van, akkor a feleslegesek figyelmen kívül maradnak, ha ke-

vesebb, akkor a hiányzó paraméterek értéke NIL lesz. A PCOUNT() függvénnyel meg tudjuk határozni az utolsó argumentum sorszámát.

Az argumentumok kétféleképpen adhatók meg: érték vagy hivatkozás (lényegében a változó értékét tartalmazó memóriacím) szerint. Az érték szerinti argumentum egy kifejezés vagy egy változónév megadása, amely esetben a függvény a paraméterértékeket tetszés szerint módosíthatja, a hívó eljárásba visszatérés után az eredeti érték marad meg. A hivatkozás szerinti argumentum esetében (ekkor a változó neve elé egy @ jelet kell írni) a változó értéke a hívó eljárásba való visszatérés után a függvény által módosított érték lesz! Mezőnév csak érték szerinti argumentumként adható meg!

Példák:

Egy összeg helyiértékek szerinti tagolása, illetve valutajel kiírása közvetlenül elé a Vjel függvény segítségével:

```
? Vjel( 5000 )      // "Eredmény:" $5,000.00
? Vjel( -5000 )     // "Eredmény:" ($5,000.00)
FUNCTION Vjel( osszeg )
   LOCAL karszam
   IF osszeg  0
karszam := TRANSFORM(-1 * osszeg, "999,999,999.99")
karszam := PADL("($" + LTRIM(karszam) + ")", LEN(karszam))
   ELSE
karszam := TRANSFORM(osszeg, "999,999,999.99")
karszam := PADL("$" + LTRIM(karszam), LEN(karszam))
   ENDIF
RETURN karszam
```

Egy felsorolásból (amelynek részeit vesszővel választottuk el), egy tömböt hozunk létre, amelynek elemei a felsorolás részei lesznek:

```
lista := Tomblist("Egy,Hat,Tiz")  // vissza:
("Egy","Hat","Tiz")
FUNCTION Tomblist( karlist )
   LOCAL hely
   LOCAL lista := {}      // "üres tömb"
   //
   DO WHILE (hely := AT(",", karlist)) != 0
AADD(lista, SUBSTR(karlist, 1, hely - 1))   // "új elem"
karlist := SUBSTR(karlist, hely + 1)
   ENDDO
   AADD(lista, karlist)
   //
RETURN lista      // "a tömb visszaadása"
```

A NIL érték át- és visszaadása, illetve annak ellenőrzése:

```
FUNCTION Valami( par1, par2, par3 )
   IF par2 != NIL
      par2 := "értékes adat"
   ENDIF
```

```
   . <utasítások>
   .
RETURN NIL
```

Példa a függvényhívás másik módjára: az első példában ismertetett Vjel függvényt egy másik munkaterületről hívjuk

```
USE Sue NEW
USE Mary NEW
? Sue->(Vjel( ertek ))
```

Lásd még: LOCAL, PARAMETERS, PRIVATE, PROCEDURE, PUBLIC, RETURN, STATIC, PCOUNT()

IF

Feltételeknek megfelelő utasításblokk kiválasztása

IF <feltétel1>
** <utasítások>...**
[ELSEIF <feltétel2>]
** <utasítások>...**
[ELSE]
** <utasítások>...**
END[IF]

<feltételx> — logikai kifejezés. Ha értéke *igaz* (.T.), akkor a következő ELSEIF, ELSE vagy ENDIF kulcsszóig következő utasításokat hajtja végre.

ELSEIF — egymásba ágyazott IF parancs helyett, ha két lehetőségnél többet kívánunk megvizsgálni. Ezzel a megadási formával nem kell ezt az ágat külön ENDIF-fel lezárni.

ELSE — erre az ágra akkor megy, ha sem az IF, sem az ELSEIF ágak feltételei nem teljesülnek.

Az IF utasítás sorban értékeli ki az IF, illetve az ELSEIF ágakon megadott logikai feltételeket, mindaddig, amíg egy *igaz* (.T.) értéket nem talál. Ha talált ilyen értéket, akkor az ott következő utasításokat végrehajtja, majd az ENDIF után következő utasításra lép át. Ha nem talált egyetlen *igaz* értéket sem, akkor az ELSE utáni utasításokat hajtja végre (ha ilyen van). Ha nincs ELSE ág, akkor ez utóbbi esetben a program az ENDIF utáni utasítással folytatódik.

Az IF...ENDIF utasításba tetszőleges számú IF...ENDIF, illetve más ciklus ágyazható.

Az IF...ELSEIF...ENDIF teljesen azonos a DO CASE...ENDCASE utasítással. Funkciója megegyezik az IF() függvénnyel, amely kifejezésként is alkalmazható.

Példa:

Az IF...ELSEIF...ENDIF szerkezet tipikus alkalmazása:

```
LOCAL szam := 0
//
IF szam  86
   ? "86 elötti"
ELSEIF szam = 86
   ? "86-os"
```

```
ELSE
   ? "86 utáni"
ENDIF
```

Lásd még: BEGIN SEQUENCE, DO CASE, DO WHILE, FOR, IF()

LOCAL

Local változók és tömbök deklarálása és kezdeti érték adása

LOCAL <azonosító> [[:= <kezdőérték>], ...]

<azonosító> — a deklarált változó vagy tömb neve. Ha az azonosítót szögletes zárójelek ([]) követik, akkor tömbként értelmezi. Tömb esetében az egyes dimenziók elemeinek maximális számát kell megadni: tomb[<dim1elem>,<dim2elem>,...<] vagy tomb[<dim1elem>][<dim2elem>]... formában. A maximális elemek száma dimenziónként 4096. A dimenziók, illetve a *local* változók maximális számát csak a gép memóriája korlátozza.

<kezdőérték> — Az új *local* változó kezdő értéke (megadása nem kötelező). Tömbök számára csupán annak nevét megadva a <kezdőérték> alkalmazásával nem tudunk azonos értékeket adni (de az egyes tömbelemek számára egyenként igen). Használata esetén az értékadó (:=) operátort kell alkalmazni, amit tetszőleges kifejezés (beleértve a karaktersorozatot és a függvényhívást is) követ. Ha nem alkalmazzuk, akkor a változó a típusának megfelelő NIL értéket veszi fel. Tömb esetében minden elem kezdőértéke NIL lesz.

Fontos! A LOCAL utasításban nem alkalmazhatunk makró (&) változót vagy makrókifejezést!

A LOCAL egy deklaráló utasítás, melynek segítségével egy vagy több változó, illetve tömb definiálható az aktuális eljárás, vagy felhasználói függvény számára. Alkalmazása esetén meg kell előznie minden egyéb végrehajtható utasítást, beleértve az esetleges PRIVATE, PUBLIC és PARAMETERS utasításokat is. Ha a *local* ként deklarált változóval azonos nevű *private* vagy *public* változó is érvényben van, akkor a *local* deklaráció érvényesül. Ugyanakkor, ha a FIELD, MEMVAR és STATIC deklarációk valamelyikével érvényben levő változónévvel azonos kerül deklarálásra, akkor a fordítóprogram lekezelhetetlen hibával leáll, és a tárgykódot (.OBJ állomány) nem hozza létre.

A *local* változók érvényessége csak arra az eljárásra vagy felhasználói függvényre terjed ki, amelyben definiáltuk. Ha az eljárás vagy felhasználói függvény rekurzív módon önmagát hívja, minden rekurzív hívás új *local* változókészletet állít elő.

Fontos tudnivalók:

- ha a *local* változók értékeit a *Clipper Debugger* alkalmazásával kívánjuk nyomon követni, akkor a fordítást a /B paraméterrel kell végezni;
- a *local* paraméterek a FUNCTION vagy a PROCEDURE neve utáni paraméterlistában történő felsorolásukkal deklarálódnak. Az így deklarált *local* paraméterek felülírják a PARAMETERS utasításban esetlegesen megadott azonos nevű *private* paraméterdefiníciót;
- *local* változót nem deklarálhatunk makróváltozón vagy -kifejezésen keresztül. Ha a makróváltozón keresztül kapott névnek megfelelő *private* vagy *public* változó lett

deklarálva, akkor azt veszi figyelembe. Ha nincs ilyen változó, akkor hibajelzéssel a program futása leáll;

- a *local* változókat nem lehet a SAVE paranccsal memóriaállományba (.mem) menteni, illetve onnan a RESTORE paranccsal visszatölteni;
- a *local* és *static* változók típusát a VALTYPE() függvény segítségével határozhatjuk meg (a TYPE() függvény ezeknél nem alkalmazható).

Példák:

Két tömb és két változó egyidejű deklarálása:

```
LOCAL tomb1[14, 17], tomb2[14] [17], valt1, valt2
```

A kezdőérték definiálása változónak, illetve tömbnek:

```
LOCAL mikor := DATE()
LOCAL tomb := {"Sue", "Mary", Lise"}
```

Lásd még: FUNCTION, PARAMETERS, PRIVATE, PROCEDURE, PUBLIC, STATIC

MEMVAR

Private és public változók deklarálása

MEMVAR <változólista>

<változólista> — a *public* és *private* változók felsorolása.

A MEMVAR utasítással a fordítóprogram számára adjuk meg a memóriaváltozóként alkalmazni kívánt azonosító neveket.

A MEMVAR utasítás nem hozza létre a változót, csak igazolja annak létezését. Ezért az itt megadott változókat a PRIVATE, PUBLIC vagy PARAMETERS utasítások valamelyikével létre kell hozni. Ha ezen utasítások megadása előtt fordulunk valamelyik MEMVAR-ban definiált változóhoz, az hibajelzéshez vezet.

A MEMVAR deklaráció hatóköre arra az eljárásra vagy függvényre terjed ki, amelyben azt megadtuk, vagy az egész forrásállományra, ha egy FUNCTION vagy PROCEDURE utasításban után adtuk meg, és a fordítást a /N paraméterrel végezzük.

A MEMVAR utasításnak meg kell előznie minden végrehajtható utasítást, beleértve a PARAMETERS, PRIVATE és PUBLIC utasítást is.

A MEMVAR kombinálható a fordítóprogram /W paraméterével, amely figyelmeztető üzenetet ad a fordítás idején deklarálatlannak tekintett változók nevéről.

Példa:

Példa arra, hogy azonos nevű változó és mező esetén mikor melyiket veszi figyelembe:

```
FUNCTION Mikormi
   MEMVAR osszeg, cim
   PRIVATE osszeg := 5000
   USE Kiadas NEW
   //
   ? osszeg                  // "a private valtozó értéke"
```

```
   ? Kiadas->osszeg     // "a Kiadas állomány összeg mezöje"
   //
RETURN NIL
```

Lásd még: FIELD, LOCAL, PRIVATE, PUBLIC, STATIC

PARAMETERS

Private paraméterváltozók létrehozása

PARAMETERS <változólista>

<változólista> — egy vagy több, vesszővel elválasztott, paraméterváltozó neve. A változók számának nem kell megegyezni a hívó eljárásban vagy felhasználói függvényben megadott argumentumok számával.

A PARAMETERS utasítás *private* változókat hoz létre az átadott értékek vagy hivatkozások fogadására. Hatására a paraméterlista összes változója *private* státuszú lesz, és az összes azonos nevű érvényes *public* és *private* változódeklaráció az illető eljárás vagy felhasználói függvény élettartama alatt felfüggesztődik. A PARAMETERS utasítás végrehajtható utasításnak számít, így az tetszőleges helyen kiadható, de a FIELD, LOCAL, MEMVAR és STATIC utasításokat nem előzheti meg!

A paraméterek *local* változóként is deklarálhatók, ha azt a PROCEDURE vagy a FUNCTION utasítás sorában, a nevet követő zárójelben adjuk meg. A paraméterdeklaráció e két formája azonban egyidejűleg nem alkalmazható, ez olyan súlyos fordítási hibát eredményez, aminek következtében a tárgymodul (.OBJ) nem jön létre.

A Clipperben az argumentumok és paraméterek számának nem kell megegyeznie. Ha a paraméterek száma kevesebb, akkor a felesleges argumentumok figyelmen kívül maradnak, ellenkező esetben pedig a hiányzó paraméterek a NIL értéket kapják. A PCOUNT() függvénnyel lekérdezhető az utolsó argumentum sorszáma. (Ha az argumentumlistában egymást követő vesszőkkel hagyunk ki értéket, az az illető paraméternél „értékes" NIL-ként lesz figyelembe véve!)

Példák:

A felhasználói függvény paramétereit a PARAMETERS utasítással definiáljuk:

```
FUNCTION Parpelda
   PARAMETERS egy, ket, har
   ? egy, ket, har
RETURN NIL
```

Az előző példa, azzal a különbséggel, hogy a paraméterek automatikusan *local* változóként deklarálódnak:

```
FUNCTION Parpelda( egy, ket, har )
   ? egy, ket, har
RETURN NIL
```

Lásd még: FUNCTION, LOCAL, PRIVATE, PROCEDURE, PUBLIC, STATIC, PCOUNT()

PRIVATE

Private memóriaváltozók és tömbök deklarálása és kezdeti értékadása

PRIVATE <azonosító> [[:= <kezdőérték>], ...]

<azonosító> — a *private* változó vagy tömb neve. Ha az azonosítót szögletes zárójelek ([]) követik, akkor tömbként értelmezi. Tömb esetében az egyes dimenziók elemeinek maximális számát kell megadni: tomb[<dim1elem>,<dim2elem>,...] vagy tomb[<dim1elem>][<dim2elem>]... formában. A maximális elemek száma dimenziónként 4096. A dimenziók maximális számát csak a gép memóriája korlátozza. A *private* és *public* változók és tömbök maximális száma egy programban 2048 lehet.

<kezdőérték> — az új *private* változó kezdő értéke (megadása nem kötelező). Tömbök számára csupán annak nevét megadva a <kezdőérték> alkalmazásával nem tudunk azonos értékeket adni (de az egyes tömbelemek számára egyenként igen). Használata esetén az értékadó (:=) operátort kell alkalmazni, amit tetszőleges kifejezés (beleértve a karaktersorozatot és a függvényhívást is) követ. Ha nem alkalmazzuk, akkor a változó a típusának megfelelő NIL értéket veszi fel. Tömb esetében minden elem kezdőértéke NIL lesz.

Egymástól vesszővel elválasztva több *private* változó és tömb deklarálható egyidejűleg egy PRIVATE utasítással.

A PRIVATE utasítás segítségével olyan változókat deklarálhatunk, amelyek érvényessége az aktuális, illetve az ez alatt elhelyezkedő eljárásokra és felhasználói programokra terjed ki. A *private* változók megszünnek az eljárás befejezésekor, vagy egy CLEAR ALL, CLEAR MEMORY, illetve RELEASE parancs kiadása esetén. Ha *private* változót deklarálunk, akkor az érvényes azonos nevű *private* és *public* változók az eljárás vagy a felhasználói függvény befejezéséig rejtve maradnak.

Ha a PRIVATE deklarációban megadottal azonos nevű változó szerepel egy előző FIELD, LOCAL vagy STATIC deklarációban, az súlyos fordítási hibát eredményez.

A PRIVATE utasítás tetszőleges helyen kiadható, de a FIELD, LOCAL, MEMVAR és STATIC utasításokat nem előzheti meg.

A PRIVATE utasítás használata mellett *private* változók még két másik úton is létrehozhatók:

- ha a változót semmilyen formában nem deklaráljuk, illetve az adott helyen azok nem érvényesek;
- paraméterátvételnél a PARAMETERS utasítás sorában megadott változók *private* státuszt vesznek fel.

Fontos! A PRIVATE utasítás számára más dBase verziókban kiadott ALL, LIKE és EXCEPT nem érvényes!

Példák:

Változók és tömbök egyidejű deklarálása:

```
PRIVATE tomb1[14], tomb2[17], valt1, valt2, valt3
```

A tömbök kétféle dimenzionálási lehetősége:

```
PRIVATE tomb1[86] [5] [30], tomb2[86, 5, 30]
```

Tömbök és változók különböző kezdőérték megadási lehetőségei a PRIVATE utasításban:

```
PRIVATE tomb1 := { 1, 2, 3, 4 }, tomb2 := ARRAY(14, 17)
PRIVATE karvalt := SPACE(17), szin := SETCOLOR()
```

Lásd még: FIELD, LOCAL, MEMVAR, PARAMETERS, PUBLIC, STATIC

PROCEDURE

Eljárás és formális paramétereinek leírása

[STATIC] PROCEDURE <eljárásnév>[(<paraméterlista>)]
[FIELD <azonosító lista> [IN <aliasnév>]]
[LOCAL <azonosító> [[:= <kezdőérték>], ...]]
[MEMVAR <azonosító lista>]
[STATIC <azonosító> [[:= <kezdőérték>], ...]]
.
. <végrehajtandó utasítások>
.
[RETURN]

<eljárásnév> — az eljárás neve. Tetszőleges hosszú lehet, de csak az első tíz karaktert veszi figyelembe; betűket, számokat és az aláhúzásjelet tartalmazhatja, azonban az aláhúzásjel az első karakter helyén más célra lett lefoglalva.

<paraméterlista> — egy vagy több paraméterváltozó felsorolása. Az itt felsorolt változók *local* deklarációt kapnak.

STATIC PROCEDURE — egy olyan eljárást deklarál, amely csak abból a program (.prg) állományból érhető el, amelyik azt tartalmazza.

LOCAL — az aktuális eljárásban érvényes változók és tömbök deklarálása és igény szerint kezdőértékkel ellátása.

STATIC — azoknak a változóknak és tömböknek a deklarálása (és igény szerint kezdőértékkel ellátása), amelyek az adott eljáráson kívül a program más részein is érvényesek.

FIELD — a mezőnévként használt azonosítók listája. Az IN kulcsszó alkalmazásával megadható annak az adatbázisnak a neve (alias neve), amely a deklarált mezőt tartalmazza. Egyébként mindig a hívatkozás helyén aktuális munkaterületen megnyitott adatbázis mezőneveként értelmezi.

MEMVAR — a *private* vagy *public* memóriaváltozók vagy tömbök deklarálása.

RETURN — az eljárásból való kilépést jelzi, és a hívó programnak, vagy felhasználói függvénynek adja vissza a vezérlést. Ha nincs megadva, akkor a visszatérés az eljárásdefiníció végén történik. Az eljárásdefiníció végét jelenti egy másik PROCEDURE, vagy egy FUNCTION utasítás, illetve egy állomány-vége jel.

A PROCEDURE utasítás deklarálja az eljárást, és igény szerint a paraméterlista megadásával a hívó program által átadott értékeket fogadó *local* változókat. Az eljárás egy alprogram, amely a függvényhívással megegyező módon az <eljárásnév> megadásával és az azt követő nyitó-záró zárójelekkel, vagy a DO ... WITH parancs alkalmazásával hívható meg. Az eljárásdefiníció a PROCEDURE kulcsszóval kezdődik, és egy másik PROCEDURE, FUNCTION

vagy RETURN kulcsszóig, illetve állomány-vég jelig tart. Ebből következően több eljárásdefiníció nem ágyazható egymásba, de egyik eljárásból egy másik (vagy rekurzív módon önmaga is) hívható.

A Clipperben az eljárás a felhasználói függvénnyel megegyező, azzal a különbséggel, hogy ez minden esetben NIL értéket ad vissza.

Az eljárások az elérhetőség szempontjából két csoportra oszthatók: a programrendszer tetszőleges helyéről hívható eljárások a PROCEDURE kulcsszóval, míg azok az eljárások, amelyek csak az azt tartalmazó programból érhetők el — a STATIC PROCEDURE kulcsszavakkal kezdődnek. A kétféle eljárás között igen lényeges különbség, hogy a STATIC eljárás a fordítás idején, míg a másik a szerkesztés fázisában illeszkedik a programrendszerhez.

Fontos tudnivalók:

Az eljárás hívása

Az eljárás hívása az alábbi módokon lehetséges:

- a javasolt módszer a függvényhívási konvenció szerinti:

<eljárásnév>([<argumentumlista>])

- a másik módszer a parancshívási konvenció szerinti:

DO <eljárásnév> [WITH <argumentumlista>]

- az eljárás meghívható alias előtag megadásával a függvényhívási konvenció szerint (ekkor az eljárásra vonatkozó részt zárójelbe kell tenni:

<aliasnév>->(<eljárásnév>([<argumentumlista>]))

Ez utóbbi esetben az eljárás érvényességi ideje alatt a munkaterület az alias névvel megadott lesz — lefutása után az eredeti munkaterület lép érvénybe.

A paraméterek

Az eljárás paramétereket tud átvenni a hívó eljárásból, felhasználói függvényből, vagy a DOS-tól. Kétféleképpen adható meg: vagy a paraméterlistában, vagy a PARAMETERS utasítás után, de a kétféle megadási mód nem keverhető — ez a fordítás során lekezelhetetlen hibát eredményez!

Az eljárás a paraméterértékeket az argumentumlistában megadott sorrendben veszi figyelembe. A megadott értékek számának nem kell megegyezni: ha több argumentumérték lett megadva, mint ahány paraméter van, akkor a feleslegesek figyelmen kívül maradnak, ha a kevesebb, akkor a hiányzó paraméterek értéke NIL lesz. A PCOUNT() függvénnyel meg tudjuk határozni az utolsó argumentum sorszámát.

Az argumentumok kétféleképpen adhatók meg: érték vagy hivatkozás (lényegében a változó értékét tartalmazó memóriacím) szerint. Az érték szerinti argumentum egy kifejezés vagy egy változónév megadása, amely esetben az eljárás a paraméterértékeket tetszés szerint módosíthatja, a hívó eljárásba visszatérés után az eredeti érték marad meg. A hivatkozás szerinti argumentum esetében (ekkor a változó neve elé egy @ jelet kell írni) a változó értéke a hívó eljárásba való visszatérés után az eljárás által módosított érték lesz! Mezőnév csak érték szerinti argumentumként adható meg! Mezőnév átadása esetén azt zárójelbe kell tenni, és elé a FIELD előtagot kell illeszteni, ellenkező esetben hibajelzés következik be!

Példák:

Egy tipikusnak nevezhető eljárásfelépítés:

```
PROCEDURE Pelda( neve, osztaly, tel, iroda )
   LOCAL fizetes, beo := {"eloado", "foeloado"}
   STATIC szaml := 0
   .
   . <utasítások> ...
   .
RETURN
```

Az argumentumlistában csak vesszővel jelzett, vagy egyéb módon NIL értékkel ellátott paraméterérték ellenőrzése:

```
PROCEDURE Ures( valt1, valt2, valt3 )
   IF valt2 != NIL
      valt2 := "adat van"
   ENDIF
   .
   . <utasítások> ...
   .
RETURN
```

Ha az eljárást nem az aktuális munkaterületről hívjuk:

```
USE Sue NEW
USE Lise NEW
Sue->(Eljar(ertek + ertek * szazalek))
```

Lásd még: FUNCTION, LOCAL, PARAMETERS, PRIVATE, PUBLIC, RETURN, STATIC, PCOUNT()

PUBLIC

A public memóriaváltozók és tömbök létrehozása és kezdőértékkel ellátása

PUBLIC <azonosító> [[:= <kezdőérték>], ...]

<azonosító> — a *public* változó vagy tömb neve. Ha az azonosítót szögletes zárójelek ([]) követik, akkor tömbként értelmezi. Tömb esetében az egyes dimenziók elemeinek maximális számát adjuk meg: tomb[<dim1elem>,<dim2elem>,...<] vagy tomb[<dim1elem>][<dim2elem>]... formában. A maximális elemek száma dimenziónként 4096. A dimenziók maximális számát csak a gép memóriája korlátozza. A *private* és *public* változók és tömbök maximális száma egy programban 2048 lehet.

<kezdőérték> — az új *public* változó kezdő értéke (megadása nem kötelező). Tömbök számára csupán annak nevét megadva a <kezdőérték> alkalmazásával nem tudunk azonos értékeket adni (de az egyes tömbelemek számára egyenként igen). Használata esetén az értékadó (:=) operátort kell alkalmazni, amit tetszőleges kifejezés (beleértve a karaktersorozatot és a függvényhívást is) követ. Ha nem alkalmazzuk, akkor a *public* változó a hamis (.F.) értéket veszi fel (a szokásos NIL érték helyett). Tömb esetében minden elem kezdőértéke itt is NIL.

Egymástól vesszővel elválasztva több *public* változó és tömb deklarálható egyidejűleg egy PUBLIC utasítással.

A PUBLIC utasítás segítségével olyan változókat deklarálhatunk, amelyek érvényessége a program összes eljárására és felhasználói függvényére kiterjed. A *public* változók érvényessége csak a program befejezésekor, vagy egy CLEAR ALL, CLEAR MEMORY, illetve RELEASE parancs kiadása esetén szünik meg. Ha a *public* változóval azonos nevű *private* , *local* vagy *static* változót vagy tömböt deklarálunk, akkor a *public* deklaráció mindaddig felfüggesztésre kerül, amíg az egyéb deklaráció érvényessége meg nem szűnik. Ugyanakkor, ha egy már érvényes *private* változóval azonos nevű *public* változót deklarálunk, akkor ez utóbbi figyelmen kívül marad (lásd a fontos tudnivalókat).

Ha a PUBLIC deklarációban megadottal azonos nevű változó szerepel egy előző FIELD, LOCAL vagy STATIC deklarációban, az súlyos fordítási hibát eredményez.

A PUBLIC utasítás tetszőleges helyen kiadható, de a FIELD, LOCAL, MEMVAR és STATIC utasításokat nem előzheti meg.

Fontos tudnivalók:

- ha *clipper* néven egy különleges public változót deklarálunk, akkor az — a többi public változótól eltérően — igaz (.T.) értéket vesz fel. Ennek segítségével olyan programot lehet írni, amely nemcsak Clipper kódot, hanem dBase III Plus vagy FoxBase utasításokat is tartalmaz (lásd példa).
- nem lehet olyan *public* tömböt deklarálni, amely névvel már létezik public vagy private változó.

Példák:

Tömbök és változók deklarálási módjai:

```
PUBLIC tomb1[14, 17], valt1
PUBLIC tomb2[14] [17]
```

A deklarálás és a kezdeti értékadás egyidejű megadása:

```
PUBLIC megj := SPACE(30), szin := SETCOLOR()
PUBLIC tomb1 := {55, 10, 14}, tomb2 := ARRAY(5, 30)
```

A speciális *clipper* változó alkalmazása:

```
PUBLIC clipper
@ 23, 10 SAY "Nyomj le egy billenty t"
IF clipper
   valasz = INKEY(0)
ELSE
   valasz = INKEY()
   DO WHILE valasz = 0
      valasz = INKEY()
   ENDDO
ENDIF
```

Lásd még: LOCAL, MEMVAR, PARAMETERS, PRIVATE, STATIC

RETURN

Az eljárás, felhasználói függvény vagy program befejezése

RETURN [<kifejezés>]

<kifejezés> — tetszőleges típusú kifejezés, a felhasználói függvény visszaadott értékének továbbítására.Ha a felhasználói függvény nem a RETURN utasításon keresztül ér véget, akkor a NIL értéket adja át.

A RETURN visszatérést eredményez a hívó eljárásba, vagy felhasználói függvénybe (a főprogram esetében az operációs rendszerbe). A hívó eljárásba való visszatéréssel egyidejűleg az eljárásban, vagy felhasználói függvényben deklarált *private* és *local* változók törlődnek.

Az eljárásban, illetve felhasználói függvényben egynél több RETURN utasítás is lehet, de a felhasználói függvényben mindin alkalommal <kifejezés>-sel együtt kell szerepelnie.

Fontos! Az eljárás vagy a felhasználói függvény végét jelzi egy új PROCEDURE, vagy egy FUNCTION utasítás, illetve az állomány-vége jel is!

Fontos tudnivalók:

- a RETURN utasítással felhasználói függvényből tömb is visszaadható;
- a Clipper nem támogatja a RETURN TO MASTER vagy más hasonló jellegű visszatérési lehetőséget megadott szintre. Ugyanakkor ez szimulálható a BEGIN SEQUENCE ... END utasítással.

Példák:

A RETURN utasítás két formája:

```
PROCEDURE <eljarasnev>
   //
   <utasítások> ...
   //
RETURN
FUNCTION <függvénynév>
   //
   <utasítások> ...
   //
RETURN <kifejezés>
```

Példa tömb visszaadására felhasználói függvényből:

```
FUNCTION Tombvissza
   PRIVATE tomb[14] [17]
   tomb[1] [1] = "egytag"
RETURN tomb
```

Lásd még: BEGIN SEQUENCE, FUNCTION, LOCAL, PRIVATE, PROCEDURE, PUBLIC, QUIT

STATIC

Static változók és tömbök deklarálása és kezdeti értékadása

STATIC <azonosító> [[:= <kezdőérték>], ...]

<azonosító> — a deklarált változó vagy tömb neve. Ha az azonosítót szögletes zárójelek ([]) követik, akkor tömbként értelmezi. Tömb esetében az egyes dimenziók elemeinek maximális számát kell megadni: tomb[<dim1elem>,<dim2elem>,...<] vagy tomb[<dim1elem>][<dim2elem>]... formában. A maximális elemek száma dimenziónként 4096. A dimenziók, illetve a *static* változók maximális számát csak a gép memóriája korlátozza.

<kezdőérték> — az új *static* változó kezdő értéke (megadása nem kötelező). Tömbök számára csupán annak nevét megadva a <kezdőérték> alkalmazásával nem tudunk azonos értékeket adni (de az egyes tömbelemek számára egyenként igen). Használata esetén az értékadó (:=) operátort kell alkalmazni, amit tetszőleges kifejezés (beleértve a karaktersorozatot és a függvényhívást is) követ. Ha nem alkalmazzuk, akkor a változó a típusának megfelelő NIL értéket veszi fel. Tömb esetében minden elem kezdőértéke NIL lesz.

Fontos! A STATIC utasításban nem alkalmazhatunk makró (&) változót vagy makrókifejezést!

A STATIC egy deklaráló utasítás, melynek segítségével egy vagy több változó, illetve tömb definiálható, amelyek érvényessége attól a pillanattól kezdődik, amikor azt megadtuk. Ha a STATIC utasítást az első eljárás, vagy felhasználó függvény hívása előtt a főprogramban adtuk ki, akkor az így deklarált változók és tömbök a teljes programrendszerre érvényesek. Ha egy PROCEDURE vagy FUNCTION fejrészében szerepel, akkor az ott deklarált változók csak az illető eljárásban vagy függényben érvényesek. A fordítóprogram /N paraméterének megadásával a változók a program egészére kiterjedő hatókörrel fognak rendelkezni.

A program static változói számára memóriaterület az első hívatkozáskor lesz lefoglalva, <kezdőérték> megadása esetén az értékadás azonnal végrehajtódik.

Alkalmazása esetén meg kell előznie minden egyéb végrehajtható utasítást, beleértve az esetleges PRIVATE, PUBLIC és PARAMETERS utasításokat is. Ha a *static* -ként deklarált változóval azonos néven FIELD, LOCAL vagy MEMVAR deklarációt is megadunk, akkor a fordítóprogram lekezelhetetlen hibával leáll, és a tárgykódot (.OBJ állomány) nem hozza létre.

Fontos tudnivalók:

- ha a *static* változók értékeit a *Clipper Debugger* alkalmazásával kívánjuk nyomon követni, akkor a fordítást a /B paraméterrel kell végezni;
- *static* változót nem deklarálhatunk makróváltozón vagy -kifejezésen keresztül. Ha a makróváltozón keresztül kapott névnek megfelelő *private* vagy *public* változó lett deklarálva, akkor azt veszi figyelembe. Ha nincs ilyen változó, akkor hibajelzéssel a program futása leáll;
- a *static* változókat nem lehet a SAVE paranccsal memóriaállományba (.mem) menteni, illetve onnan a RESTORE paranccsal visszatölteni;
- a *local* és *static* változók típusát a VALTYPE() függvény segítségével határozhatjuk meg (a TYPE() függvény ezeknél nem alkalmazható).

Példák:

A *static* változók deklarálása kezdeti értékadással és anélkül:

```
STATIC tomb1[10, 14], tomb2[10] [14]
STATIC valt1, valt2
STATIC szov := "valami", valt3
STATIC tomb3 := {86, 5, 30}
```

Példa arra, hogyan lehet a *static* változó értékét manipulálni felhasználói függvényben, ha nekünk arra a függvény többszöri meghívása után szükség van, például nyomtatásnál a fejlécben a lapszám folyamatos írásához:

```
FUNCTION Fejlec( ujlap )
   STATIC lapsz := 0      // "az elsö híváshoz"
   IF ujlap != NIL
      lapsz = ujlap       // "az elözö érték"
   ENDIF
      lapsz++             // "az új érték"
   <fejlec nyomtatas>
   RETURN lapsz
```

Példa a teljes hatókörű deklarálásra, a tömb értékei minden függvényben érvényesek:

```
STATIC tomb := {1, 2, 3, 4}
FUNCTION Egy
   ? tomb[1]             //Eredmény: 1
RETURN NIL
FUNCTION Ket
   ? tomb[3]             //Eredmény: 3
RETURN NIL
```

Lásd még: FUNCTION, LOCAL, PARAMETERS, PRIVATE, PROCEDURE, PUBLIC

Előfordító (pre-processzor) direktívák

#command/#translate

Felhasználó által definiált parancs- vagy fordítási direktíva specifikálása

#command <mintaalak> => <eredményalak>
#translate <mintaalak> => <eredményalak>

<mintaalak> — az input szövegben az átalakításra kerülő rész.

<eredményalak> — az átalakított alak, ha az input szövegben előfordul a <mintaalak>.

A => jel a <mintaalak> és az <eredményalak> között a szintaktika része, amit kötelező megadni. Nem szabad összetéveszteni a Clipper >= vagy <= relációs operátoraival.

A #command és a #translate fordítási direktívák parancsok és pszeudo-függvények definiálására szolgálnak. Minden direktívát egy fordítási szabály határoz meg, amelynek két része a <mintaalak> és az <eredményalak>.

A #command és a #translate hasonló funkciót töltenek be, a különbség annyi, hogy a #command direktíva az illesztést csak teljes utasítás esetén végzi el, míg a #translate részutasítások esetében is. Ennek megfelelően a #command egy komplett parancs, míg a #translate pszeudo-függvények létrehozására használható.

A #command és a #translate hasonlítanak a #define direktívára, azonban annál általánosabbak.

A #command és a #translate direktívák hatóköre megegyezik a #define hatókörével.A definíció csak az aktuális programban (.prg) állományban érvényes, kivéve ha az az STD.CH-ban történt, vagy ha a fordítási parancshoz a /U paramétert is megadtuk. Ha nem az STD.CH-ban lett definiálva, és a /U paraméter sem lett megadva, akkor a definíció a megadás helyétől a programállomány végéig érvényes. A #define-nal ellentétben, sem a #command sem a #translate definíciót nem lehet explicit módon definiálatlanul hagyni. Az #undef direktíva nincs hatással a #command és a #translate definíciókra.

A Clipper 5.0 minden parancsa definiálva lett a #command direktívával kifejezések és függvényhívások alakjában, ezeket az INCLUDE alkönyvtár STD.CH állománya tartalmazza. Ezzel a megoldással biztosították a korábbi dBase és Clipper verziókkal való kompatibilitást. Egy parancsra ilyenformán több #command definíció is vonatkozhat, de ugyanígy különböző sorokban definiáljuk például a GET parancs VALID, illetve RANGE paraméteres formáját is.

A mintaalak

A fordítási direktíva része, amelyet be kell helyettesíteni. Egy vagy több részből állhat:

- *karaktersorozat* : amely változatlan formában jelenik meg az eredményalakban;
- *kulcsszavak* és *azonosítók*: a mintaalaknak kötelezően kulcsszóval kell kezdődnie;
- *mintacímkék*: amelyeket a <mintaalak> és az <eredményalak> ugyanúgy tartalmazza. Ezek azonosítják az egyes behelyettesítésre kerülő részeket. A címkenevekneknek betűvel vagy aláhúzáskarakterrel kell kezdődnie, és alfanumerikus, valamint aláhúzáskarakterekből állhat. Minden mintacímkét a

kisebb-nagyobb (< >) jelek közé kell tenni, lehetséges formái: <címkenév>; <címkenév,...>; <mintacímke:kulcsszó>; <*mintacímke*>.

- *opcionális paraméterek*: a mintaalak szögletes zárójelek ([]) közé zárt része. A <mintaalak>-nak azt a részét jelzi, amely hiányozhat az input szövegből.

Az eredményalak

Az <eredményalak> egy vagy több részből állhat:

- *karaktersorozat* , amely változatlan formában kerül át;
- *kulcsszavak* és *azonosítók*, amelyek változatlan formában kerülnek át;
- *eredménycímkék*, amelyek a mintacímke-neveket definiálják. Lehetséges formái: <mintacímke>, #<mintacímke>, <"mintacímke">, <(mintacímke)>, <{mintacímke}>, <.mintacímke.>;
- *ismétlődő rész*: az eredménycímke szögletes zárójelek ([]) közé zárt része, amely azt mutatja, hogy az a rész többszörösen ismétlődhet.

Fontos tudnivalók:

- Ha a kisebb jel (<) az eredménycímke része, azt az escape karakternek (\) kell megelőznie, például: \<.
- az <eredménycímke> több utasítást is tartalmazhat, ezeket egymástól pontosvesszővel kell elválasztani. Mivel a pontosvesszőt alkalmazzuk akkor is, ha egy utasítás nem fér el egy sorba, ezért ha egy új utasítást új sorban kezdünk, akkor két pontosvesszőt kell kitenni.

Példák:

Az STD.CH állomány minden lehetséges formát tartalmaz, érdemes áttekinteni ezeket. Néhány tipikus alkalmazási lehetőség:

```
#command ? [<lista,...>] => QOUT(<lista>)
```

A mintacímke:kulcsszó alak használata: ha az ADDITIVE kulcsszó szerepel, akkor az .add. értéke igaz (.T.), ellenkező esetben hamis (.F.).

```
#command RESTORE FROM <file> [<add: ADDITIVE>];
=>;
   CmdRestore( <(file)>, <.add.> )
```

Egy lista tartalmának tömbbe helyezése:

```
#command COPY TO <file> [FIELDS <mezok,...>];
=>;
   CmdCopyAll( <(file)>, { <(mezok)> } )
```

A korábbi verziókban szereplő, de már nem értelmezett utasítások átlépésének lehetősége (eredményként semmit nem kívánunk látni):

```
#command SET ECHO <*szoveg*>    =>
#command SET TALK <*szoveg*>    =>
```

Példa az eredményalakban előforduló több utasításra:

```
#command COUNT [TO <valt>];
   [FOR <for>] [WHILE <while>];
   [NEXT <next>] [RECORD <rec>] [<rest:REST>] [ALL];
=>;
```

```
   <valt> := 0;;
   DBEVAL( {|| <valt>++}, <{for}>, <{while}>,;
      <next>, <rec>, <.rest.> )
```

Példa az opcionális paraméterek megadási módjára, és kezelésére:

```
#command USE <adatbazis> [<new: NEW>] [ALIAS <a>];
   [INDEX <index,...>][<ex: EXCLUSIVE>][<sh: SHARED>];
   [<ro: READONLY>];
=>;
   CmdOpenDbf(<(adatbazis)>, <(a)>, <.new.>,;
      IF(<.sh.> .OR. <.ex.>, !<.ex.>, NIL), <.ro.>,;
         {<(index)>})
```

Egy opcionális mintacímke és egy ismétlődő eredménycímke alkalmazási módja:

```
#command STORE <ertek> TO <valt1> [, <valtN> ];
=>;
   <valt1> := [ <valtN> := ] <ertek>
```

Pszeudo-függvény definiálása a #translate segítségével:

```
#translate AllTrim(<karsor>) => LTRIM(RTRIM(<karsor>))
```

Lásd még: #define

#define

Konstanshozzárendelés vagy pszeudo-függvény definiálása

#define <konstansazonosító> [<helyettesítő szöveg>]
#define <függvényazonosító>([<argumentumlista>]) [<kifejezés>]

<konstansazonosító> — a definiálásra kerülő azonosító neve.

<helyettesítő szöveg> — a helyettesítő szöveg helyesen megadott konstansazonosító esetén.

<függvényazonosító> — pszeudo-függvény definiálás (opcionális argumentumlistával). A függvényazonosítót kötelezően közvetlenül (szóköz kihagyása nélkül) egy nyitó zárójelnek kell követnie, amelynek záró párja az argumentumlista után van.

<kifejezés> — a pszeudo-függvény helyettesítő kifejezése, amelyet az argumentumlistától való jobb elkülönítés érdekében zárójelben adjuk meg.

Fontos! A #define különbséget tesz a kis- és nagybetűk között — ellentétben a #command és a #translate azonosítóival!

A #define direktíva egy azonosítót definiál, amelyet a megadott kifejezéssel vagy szöveggel helyettesítünk. A helyettesítéskor a kifejezésben szereplő változók aktuális értékei jelennek meg.

Az azonosítók nevére a Clipper általános előírásai vonatkoznak: alfanumerikus karakterek és az aláhúzásjel szerepelhet bennük; hosszuk tetszőleges, de csak az első tíz karaktert veszi figyelembe.

Minden #define definíció egy sorból áll, ha nem fér ki egy sorba, akkor a sor végére pontosvesszőt kell tenni.

A definíció csak abban a program (.prg) állományban érvényes, amelyben az szerepel, kivéve, ha azt az STD.CH állomány tartalmazza, illetve ha a fordítás a /U paraméterrel lett végrehajtva. A definiált azonosító a definiálás helyétől kezdve addig van érvényben, míg egy #undef direktíva azt meg nem szünteti; egy másik #define paranccsal át nem írjuk; illetve a program véget nem ér.

A #define direktíva három alapvető felhasználási területe:

- ellenőrző azonosító definiálása az #ifdef, illetve #ifndef direktívákhoz;
- egy azonosítóhoz konstans értéket rendelünk;
- egy fordítási pszeudo-függvény definiálása.

Előfordító azonosítók

A leggyakoribb alkalmazási forma, amikor nem adunk meg helyettesítő szöveget, hanem az azonosítót egy későbbi #ifdef vagy #ifndef direktívával ellenőrízzük. Akkor használjuk, ha valamely azonosítót a fordításban szerepeltetni akarjuk, vagy nem akrjuk szerepeltetni (feltételes fordítás). Az ilyen típusú azonosítók a fordítóprogram /D paraméterével is definiálhatók.

Konstanshozzárendelés

Ezzel az alkalmazással egy változóhoz konstans értéket rendelhetünk. Gyakori formája, amikor a hozzárendelt érték az INKEY() kód, például:

```
#define K_ESC    27
IF LASTKEY() = K_ESC
.
. <utasítások>
.
ENDIF
```

A konstanshozzárendelésnek számos előnye van:

- a fordítóprogram jóval gyorsabban működik a konstansok alkalmazása esetén, mint változók esetében;
- a változók memóriaterületet foglalnak, míg a konstansok nem — ezzel memória takarítható meg;
- egy konstansérték változóban őrzése veszélyeket hordozhat magában: a változó értéke véletlenül felülíródhat, míg a konstanssal ez nem fordulhat elő;
- a konstans értékek definiálása egy helyen végezhető el, így azok könnyeben ellenőrízhetők és azonosíthatók.

A konstansok és más azonosítók hozzárendelései egy állományba csoportosíthatók, amely állományt (header file) az #include direktívával kapcsolhatjuk a programhoz. Ilyen kapcsolódó (header) állományokat tartalmaz az INCLUDE alkönyvtár.

Fordítási pszeudo-függvények

A pszeudo-függvények kiértékelése és behelyettesítése a fordítás idején történik meg. A pszeudo-függvényben az azonosító nevét egy nyitó zárójel követi, amely után — szóköz kihagyása nélkül — következik az argumentumlista, majd a zárójel bezárását követően a helyettesítő kifejezés, például:

```
#define AREA(hossz, szel)    (hossz * szel)
#define SETVAR(x, y)         (x := z)
#define MAX(x, y)            (IF(x > y, x, y))
```

A pszeudo-függvények a konstanshelyettesítéstől az argumentumok alklamazási lehetőségével különbözik. Amikor az előfordító egy függvényhívással találkozik, először ellenőrzi, hogy erre a függvénynévre nincs-e érvényes pszeudo-függvény definíció. Ha van, akkor a függvény meghívása helyett az azonos sorban megadott kifejezést helyettesíti be. Például a fenti példák esetén a programlistában megadott:

```
? AREA(10, 12)
SETVAR(ertek, 10)
? MAX(10, 9)
```

helyett a fordítóprogram a következőkkel dolgozik:

```
? (10 * 12)
(ertek := 10)
(IF(10 > 9, 10, 9))
```

Nagyon fontos, hogy a helyettesítő kifejezéseket zárójelbe tegyük, különösen a numerikus értékek esetében. A pszeudo-függvényeknél — ellentétben a felhasználó által definiált függvényekkel és eljárásokkal — minden argumentumot specifikálni kell, ellenkező esetben hibajelzést kapunk.

A pszeudo-függvények a felhasználó által definiált függvényeknél jóval gyorsabb végrehajtást és kisebb memóriaterület foglalását eredményezik, hátránya ugyanakkor, hogy nehezebb a debuggerrel történő nyomkövetésük, behatárolt hatókörrel rendelkeznek, nem hagyhatók el argumentumok és különbséget tesznek a kis- és nagybetűk között (case-szenzitivitás). Ezeknek a hiányosságoknak a kiküszöbölésére a pszeudo-függvények definiálásához alkalmazzuk a #translate direktívát, amely nem tesz különbséget a kis- és nagybetűk között, argumentumai opcionálisan adhatók meg, és alkalmazkodik a dBase négybetűs szabályához (elegendő az utasítások első négy betűjét megadni).

Példák:

A konstanshelyettesítés alkalmazása a nyomkövetési kód feltételellenőrzéséhez:

```
#define DEBUG
.
. <utasítások>
.
#ifdef DEBUG
   Assert(FILE("System.dbf"))
#endif
```

Konstanshelyettesítés INKEY() értékkel:

```
#define K_ESC 27
.
. <utasítások>
.
IF INKEY() != K_ESC
   Tedd()
```

```
ELSE
   Allj()
ENDIF
```

Pszeudo-függvény definiálása a Clipper szabványfüggvényeihez:

```
#define MAX(arg1, arg2)      (IF(arg1 > arg2, arg1, arg2))
#define ALLTRIM(karsor)      (RTRIM(LTRIM(karsor)))
.
. <utasítások>
.
? MAX(1, 2)                  // 2
? ALLTRIM("   Hello   ")     // Hello
```

Lásd még: #ifdef, #ifndef, #translate, #undef

#ifdef

Programrész fordítása definiált azonosító esetén

#ifdef <azonosító>
** <utasítások>**
[#else]
** <utasítások>**
[#endif]

<azonosító> — annak az azonosítónak a neve, amelynek a definiálását ellenőrízni kell.

Az #ifdef...#endif segítségével hajtható végre, az úgynevezett **feltételes fordítás**. Segítségével meghatározható, hogy a program melyik része legyen lefordítva. A vizsgált azonosítót vagy a #define direktívával, vagy a fordítóprogram /D paraméterével definiálhatjuk.

Az #else direktíva utáni részek kerülnek befordításra, ha az <azonosító> nem lett definiálva. Az #endif direktíva zárja a feltételes fordítási blokkot.

A feltételes fordítás igen sok alkalmazási lehetőséget rejt magában. Ezt alkalmazzuk a nyomkövetésnél (lásd példa a #define direktívánál), és például igen jól alkalmazható abban az esetben, ha a DEMO változatot és az éles rendszert egy programban kívánjuk tartani.

Példák:

A feltételes fordítás egyik alapesete:

```
#define DEMO
.
. <utasítások>
.
#ifdef DEMO
   <DEMO utasítások>
#endif
```

A fordítóprogram parancssorában megadott /D paraméter használata a feltételes fordítást szabályzó változó definiálásához:

A DOS-ban:

```
C>CLIPPER Sueprog /DDEBUG
```

A programban (.prg állományban):

```
#ifdef DEBUG
   Assert(<feltétel>)
#endif
```

Az #ifdef használatának bemutatása arra az esetre, ha más konstans értéket akarunk adni egy változónak akkor, ha már volt definiálva, és mást, ha még nem:

```
#ifdef M_MARGO
   #undef M_MARGO
   #define M_MARGO  15
#else
   #define M_MARGO  10
#endif
```

Lásd még: #define, #ifndef

#ifndef

Programrész fordítása nemdefiniált azonosító esetén

#ifndef <azonosító>
** <utasítások>**
[#else]
** <utasítások>**
[#endif]

<azonosító> — annak az azonosítónak a neve, amelynek a definiálását ellenőrízni kell.

Az #ifndef...#endif segítségével ugyancsak **feltételes fordítás** hajtható végre. Akárcsak az #ifdef...#endif direktívával, ezzel is az határozható meg, hogy a program melyik része legyen lefordítva. A vizsgált azonosítót a #define direktívával, a fordítóprogram /D paraméterével, egy kapcsolódó (header) állományban, vagy a CLIPPERCMD környezeti változóval definiálhatjuk.

Az #else direktíva utáni részek kerülnek befordításra, ha az <azonosító> definiálva lett. Az #endif direktíva zárja a feltételes fordítási blokkot.

Példák:

A feltételes fordítás egyik alapesete:

```
#define DEBUG
... <utasítások>...
#ifndef DEBUG
   <a program normál változata>
#else
   <a program nyomkövetéses változata>
#endif
```

A fordítóprogram parancssorában megadott /D paraméter használata a feltételes fordítást szabályzó változó definiálásához:

A DOS-ban:

```
C>CLIPPER Sueprog /DNODEBUG
```

A programban (.prg állományban):

```
#ifndef NODEBUG
   Assert(<feltétel>)
#endif
```

A fordítóprogram parancssorában a /D paraméterrel megadott alapérték visszaállítása a programban, ha közben a definiálás megszünne:

A DOS-ban:

```
C>CLIPPER Sueprog /DM_MARGO=15
```

A programban:

```
#ifndef M_MARGO
   #define M_MARGO  15
#endif
```

Lásd még: #define, #ifdef

#include

Az aktuális állomány kiegészítése egy másik állománnyal

#include "<kapcsolódó állomány neve>"

<kapcsolódó állomány neve> — annak az állománynak a neve, amelyet az aktuális állományhoz kívánunk kapcsolni. Kötelezően dupla idézőjelek közé kell tenni. Ha elérési útvonalat nem adunk meg, akkor az előfordító a keresést a következő helyeken és sorrendben végzi:

- az aktuális könyvtárban;
- a /I paraméterrel megadott könyvtárakban;
- az INCLUDE környezeti változóval definiált könyvtárakban.

Az #include direktívák 16 szint mélységig ágyazhatók egymásba.

Az #include megadásának helyén behelyettesíti a kapcsolódó állomány tartalmát a programba. A kapcsolódó állomány tetszőleges utasításokat tartalmazhat, beleértve más előfordító direktívákat is, kiterjesztése .ch.

A gyakran alkalmazott pszeudo-függvényeket, illetve konstanshelyettesítéseket célszerű ilyen kapcsolódó állományokba összegyűjteni, amire jó példa az Inkey.ch, amely az INKEY() kódok konstans változóba helyettesítését tartalmazza. Ennek használatával könnyebben azonosíthatók az egyes billentyűk.

Az #include direktívával kapcsolt állomány abban a program (.prg állományban) érvényes, amelyben az kiadásra került, kivéve, ha a fordítóprogram parancssorában a /U paramétert adjuk meg. Ekkor az összes programra kiterjed érvényessége.

Fontos tudnivalók:

- A Clipper 5.0 verzióját bőven ellátták kapcsolódó (header) állományokkal, amelyek az INCLUDE alkönyvtárban találhatók.
- Az STD.CH a Clipper különleges szabvány kapcsolódó állománya. Ez tartalmazza a Clipper 5.0 összes parancsának és szabvány függvényének pszeudo-függvényként definiált alakját.

Példa:

Az Inkey.ch kapcsolódó állomány használata az egyes billentyűk azonosító névvel való hivatkozásához:

```
#include "Inkey.ch"
FUNCTION GetNyom()
   LOCAL bill, eredm
   bill = INKEY(0)
   DO CASE
   CASE bill = K_F10
      eredm := Tedd("Browse")
   CASE bill = K_ESC
      eredm := Kilep()
   .
   . <utasítások>
   .
   CASE bill = K_CTRL_RIGHT
      eredm := Kovrek()
   ENDCASE
   RETURN eredm
```

Lásd még: #command, #define, #translate

#undef

A #define makródefiniálás megszüntetése

#undef <azonosító>

<azonosító> — a megszüntetendő konstanshelyettesítő vagy peszeudo-függvény neve.

Az #undef megszünteti a #define hatását. Az #undef-et célszerű kiadni egy #define-nal történő újradefiniálás előtt is, ezzel kiküszöbölve a fordítóprogram figyelmeztető jelzését. Az #undef alkalmazható a feltételes fordításoknál a szükséges programág meghatározásához.

Példák:

Az #undef direktíva alkalmazási formái:

```
#define K_ESC    27
#define MAX (x, y)    IF(x > y, x, y)
.
. <utasítások>
.
```

```
#undef K_ESC
#undef MAX
```

Az #undef használata az újradefiniálás kiadása előtt:

```
#define DEBUG
.
. <utasítások>
.
#undef DEBUG
#define DEBUG .T.
```

Annak ellenőrzése, hogy egy változó definiálva lett-e már:

```
#ifdef SUE
   #undef SUE
#else
   #define SUE
#endif
```

Lásd még: #define, #ifdef, #ifndef

Clipper parancsok

?|??

Kiíratás a képernyőre/nyomtatóra

?|?? [<kifejezéslista>]

<kifejezéslista> — a kiíratandó érték(ek), amely különböző adattípusok kombinációja lehet, beleértve a memo típust is. A lista elemeit vesszővel kell elválasztani. Elhagyása esetén ? megadásánál "kocsivissza-soremelés" következik be; ?? esetén semmi nem történik.

A ? ás a ?? parancsok a QOUT(), illetve a QQOUT() függvények szinonímái.

A ? használata esetén a <kifejezéslista>-ban megadott értékek a következő sor elejétől; ?? használata esetén az aktuális kurzor (nyomtatófej) pozíciótól kerülnek kiírásra.

SET PRINTER OFF kapcsolóállásban a ROW() és a COL(); SET PRINTER ON állásban a PROW() és a PCOL() veszi fel a kurzorpozíció, illetve a nyomtatófej állásának aktuális értékét. Ha a kiírandó karaktersorozat meghaladja a képernyő jobb szélét meghatározó MAXCOL() értéket, akkor a kiírást a következő sorban folytatja; ha a képernyő alsó sorát meghatározó MAXROW() értékét haladja meg, akkor egy sor felfelé görgetését "scrollozását" hajtja végre.

A SET PRINTER ON kapcsolóállítással nyomtatóra, SET ALTERNATE TO <állománynév> definiálása után SET ALTERNATE ON megadásával állományba vezérelhető a kiíratás. A SET CONSOLE OFF állítással letiltható a képernyőn történő megjelenítés.

A kifejezéslistában használható a TRANSFORM() vagy más önállóan definiált függvény. A változó hosszúságú kifejezések elhelyezéséhez (középre, jobbra, balra) — a példában bemutatott módon — a PAD() függvények alkalmazhatók.

Példa:

Adatbázis tartalmának táblázatos formában történő nyomtatása, a táblázat fejrészének igazításával:

```
LOCAL Lap := 0, Sor := 99
USE Vevo INDEX Vevo NEW
SET PRINTER ON
SET CONSOLE OFF
DO WHILE !EOF()
   IF Sor > 55
       IF Lap != 0
          EJECT
       ENDIF
       ? PADR("Lap:", LTRIM(STR(Lap++)), 72)
       ?? DTOC(DATE())
       ?
       ?
       ? PADC("Vevölista", 79)
       ?
       Sor := 5
```

```
   ENDIF
   ? Nev, Cim, PADR(RTRIM(Varos) + ",") , Irszam
   Sor++
   SKIP
ENDDO
SET CONSOLE ON
SET PRINTER OFF
CLOSE Vevo
```

Helye: CLIPPER.LIB

Lásd még: @...SAY, SET CONSOLE, SET PRINTER, PAD(), TRANSFORM()

@...BOX

Keretrajzolás a képernyőn

@ <felső sor>, <bal oszlop>, <alsó sor>, <jobb oszlop> BOX <keretezők>

<felső sor>, <bal oszlop>, <alsó sor>, <jobb oszlop> — numerikus kifejezések, a keret négy sarokpontja. A sarokpontok akkor látszanak a képernyőn, ha a sor 0 és MAXROW(), az oszlop 0 és MAXCOL() közötti értéket vesz fel.

<keretezők> — karakteres kifejezés, a keretet alkotó jelek (9 darab) az óramutató járásával megegyező sorrendben a bal felső ponttól indulva. A 9.jel a keretet feltöltő karakter. Ha csak egyetlen karaktert adunk meg, akkor a teljes keretet ez az egy karakter alkotja.

A keret megrajzolása után a kurzor a kereten belüli bal felső pozícióba (<felső sor>+1, <bal oszlop>+1) kerül, amely értéket természetesen a ROW() és COL() függvények is átveszik.

Példák:

Az első két példa a Box.ch állományba elhelyezett keretezőjelek felhasználásával dolgozik. Az elsőben csak a keretvonalak kerülnek megrajzolásra, a másodikban a keret üres helyekkel feltöltésre (azaz törlésre) kerül.

```
#include "Box.ch"
// "Dupla vízszintes és szimpla függöleges vonalakkal határolt
keret:"
@ 1, 1, 22, 79 BOX B_DOUBLE_SINGLE
// "Az elözövel megegyezö keret üres helyekkel feltöltve"
@ 1, 1, 22, 79 BOX B_DOUBLE_SINGLE + SPACE(1)
```

A következő példában a kurzor éppen aktuális pozíciójától a képernyő jobb alsó sarkáig tartó dupla keretet rajzolunk, amelyet üres helyekkel fel is töltünk. A keretező karaktereket közvetlenül adjuk meg:

```
@ ROW(), COL(), 23, 79 BOX "ÍÍ "
```

A képernyőre egy pontrácsot rajzolunk:

```
@ 1, 1, 15, 15 BOX "."
```

Helye: CLIPPER.LIB, kész rutinok rajzoláshoz: Box.ch.

Lásd még: @...CLEAR, @...TO, SCROLL()

@...CLEAR

Képernyőablak törlése

@ <felső sor>, <bal oszlop> [CLEAR [TO <alsó sor>, <jobb oszlop>]].

<felső sor>, <bal oszlop>, <alsó sor>, <jobb oszlop> — numerikus kifejezések, a törlésre kijelölt ablak négy sarokpontja. Ha a TO nincs megadva, akkor a jobb alsó sarokpont automatikusan a MAXROW() és a MAXCOL() szerinti értékeket veszi fel.

A @...CLEAR parancs a képernyő kijelölt részét az aktuális alapszínnel feltöltve törli. A kurzor a törlésre kijelölt kereten belüli bal felső pozícióba (<felső sor>+1, <bal oszlop>+1) kerül, amely értéket természetesen a ROW() és COL() függvények is átveszik.

Példa:

A példában kék színnel feltöltjük (letöröljük) a 10,10-től 20,40 pontok által határolt területet, majd egy világos cián színű keretet rajzolunk erre a helyre:

```
SETCOLOR("BG+/B")
@ 10,10 CLEAR TO 20,40
@ 10,10 TO 20, 40
```

Helye: CLIPPER.LIB

Lásd még: @...BOX, CLEAR SCREEN, SCROLL(), SETCOLOR()

@...GET

Adatbekérés a képernyő meghatározott területén

@ <sor>, <oszlop> [SAY <kifejezés> [PICTURE <formátum1>]]
GET <változónév> [PICTURE <formátum2>] [WHEN <feltétel>]
[RANGE <alsó>, <felső>] | [VALID <log.kif.>]

<sor>, <oszlop> — numerikus kifejezések, az adat bekérésének helye a képernyőn. Ha a SAY kulcsszóval szöveget is íratunk ki, akkor a szöveg kiírásának kezdőpozíciója, a GET szerinti adatbekérés a szöveg után kezdődik.

SAY <kifejezés> — a <kifejezés> kiíratása a képernyő meghatározott helyére. Ha a hozzá tartozó PICTURE is meg van adva, akkor a kiíratás formázott lesz.

GET <változónév> — a bekért adat értékét felvevő változó neve. Típusa lehet: karakteres, dátum, numerikus vagy logikai. Ha tömb számára kérünk be értéket, a megfelelő tömbindex(ek)et is meg kell adni. A megadott érték a képernyő megfelelő helyén is kiírásra kerül.

PICTURE <formátum2> — a bekért adat kijelzésének formázása.

WHEN <feltétel> — logikai kifejezés, melynek teljesülése szükséges ahhoz, hogy a GET szerint beadott értéket a READ értelmezze.

RANGE <alsó>, <felső> — dátum vagy numerikus kifejezés, a beírásra kerülő érték lehetséges alsó és felső határa.

VALID <log.kif.> — egy olyan kifejezés, amelynek teljesülése szükséges ahhoz, hogy a beírásra kerülő érték a READ számára elfogadható legyen.

A @...GET parancs a képernyő megadott helyén bekér és kiír egy értéket, melyet átad a meghatározott változónak. A változó tetszőlegesen lehet *private* , *public* , *local* vagy *static* változó, illetve adatbázismező. Ha a változó egy vagy több indexet is tartalmaz, akkor tömbként lesz figyelembe véve.

A READ parancs a GET-ek által meghatározott értékeket "full-screen" üzemmódban veszi figyelembe, azaz a READ parancs kiadásáig a GET-ek által meghatározott mezők között tetszőlegesen mozoghatunk, a már megadott értéket módosíthatjuk stb. A READ után kerülnek a képernyőn látható értékek a változó(k) számára átadásra.

Ha a WHEN után megadott kifejezés értéke hamis (.F.), az ehhez a feltételhez kötött GET szerinti adatbekérés elmarad. Ha a VALID és a RANGE szerinti feltételek nem teljesülnek, a READ a beadott értéket nem fogadja el.

A PICTURE-ben megadott formátummal az adatbekérés (kiíratás) formázható. A <formátum> a formázó funkciót és a maszkot tartalmazhatja. A funkciót meghatározó sztring elé a @ jelet kell írni. Egyidejüleg több funkció is megadható, felsorolásukkor nem kell közöttük üres helyet hegyni. Ha a <formátum>-ban egyidejüleg adunk meg formázó funkciót és maszkot, akkor először kötelezően a funkciónak kell állnia, amelyet a maszktól egy üres hely választ el.

A GET PICTURE formázó funkciói

Funkció	Tipus	Hatás
A	C	Csak alfabetikus karaktereket fogad el
B	N	A számokat balra igazítja
C	N	A pozitív számok után CR (követel) kódot jelez
D	D,N	A dátumokat a SET DATE szerinti formában mutatja
E	D,N	Dátum típusú változóban felcseréli a napot és a hónapot. Numerikus változónál felcserélődik a tizedes pont és a helyiértéket jelző vessző funkciója
K	ALL	Ha az elsőként lenyomott billentyű nem kurzormozgató billentyű, akkor törli a mező eddigi tartalmát
R	C	A <formátum>-ban megadott nem-funkció és nem-maszk karaktereket a képernyőn megjeleníti, de a változó értékénél nem veszi figyelembe
S<n>	C	Karakterlánc bekérésekor egy egysoros ablakban dolgozik. Ha a beírni kívánt változó szerinti adat hosszabb, mint a rendelkezésre álló hely, akkor vizszintes görgetést hajt végre. <n> — egész szám, az adatmegjelenítésre szolgáló ablak hossza
X	N	A DB (tartozik) megjelenítése negatív szám után (előjelet nem ír!)
Z	N	A vezető nullák helyett üres hely (zéróelnyomás)
(	N	A negatív számokat zárójelbe teszi, a vezető üres helyeket meghagyja
)	N	A negatív számokat zárójelbe teszi, a vezető üres helyeket elhagyja
!	C	Az alfabetikus karaktereket nagybetűre konvertálja

Jelmagyarázat : C — karakteres; D — dátum; N — numerikus; ALL — minden típus

A maszkok segítségével határozhatjuk meg, hogy a beírni kívánt adatunk egyes karakterei milyen típusúak legyenek. Ha a maszkban a táblázatban megadott maszkkaraktereken kívül más karakter is szerepel, akkor az a megadott helyen a mező tartalmává válik, kivéve, ha a @R formázó karaktert is definiáltuk.

A GET PICTURE maszkkarakterei

Maszk	Hatás
A	Csak alfabetikus karaktereket fogad el
N	Csak alfabetikus és numerikus karaktereket fogad el
X	Tetszőleges karaktert elfogad
9	Csak számjegyeket (numerikus típus esetén előjelet) fogad el tetszőleges adattípusban
#	Számjegyet előjelet és üres helyet fogad el (tetszőleges adattípusnál)
L	Csak a T, F, Y és N értékeket fogadja el
Y	Csak a T, F, Y és N értékeket fogadja el
!	Az alfabetikus karaktert nagybetűre konvertálja
$	A szám elején levő nullákat $-ral tölti fel
*	A szám elején levő nullákat *-gal tölti fel
.	Tizedespont megjelenítése
,	A számok tagolása

Ha a megadott érték helytelen (például a RANGE-ben szereplő intervallumon kívül esik), akkor a képernyőn üzenet jelenik meg. Az üzenetet a SCOREBOARD terület tartalmazza, és a SET SCOREBOARD paranccsal ki-, illetve bekapcsolható.

Ha egy GET mezőből az ESC billentyű lenyomásával lépünk ki, az előző READ óta megadott GET értékek elvesznek, és a READ parancs nem kerül végrehajtásra. Ez a lehetőség a SET ESCAPE paranccsal állítható be, illetve elnyomható.

A SET KEY parancs segítségével a funkcióbillentyűk lenyomására különböző programok, eljárások aktivizálódnak. A SET KEY által meghatározott eljárás lefutása után a GET ismét érvénybe lép.

A GET-ek által meghatározott értékek mindaddig érvényben maradnak, amíg egy READ, egy CLEAR, vagy egy CLEAR GETS parancs kiadásra nem kerül.

Ha a SET DELIMITER kapcsoló ON állapotban van, a SAY utáni kifejezés és a GET utáni érték közé az aktuális DELIMITER (elválasztó) karakter kerül. Ha az INTENSITY kapcsoló ON állapotban van, az éppen megadásra kerülő GET intenzívebb színnel látszik. INTENSITY OFF esetben minden GET mező azonos intenzítású.

A GET értékének megjelenítendő hosszát az azonosító hossza, a PICTURE-ben megadott formátum, illetve maszk határozza meg. Ha az @S funkciót megadtuk, akkor a képernyőn az <n> szerinti hossz látható.

A WHEN és a VALID kifejezése helyén külső funkciók is állhatnak, amelyek automatikusan végrehajtásra és kiértékelésre kerülnek a GET aktivizálása során. A SET KEY segítségével GET-specifikus help készíthető, ha a SET KEY által meghatározott eljáráson belül a READVAR() függvényt használjuk.

Példák:

A VALID kulcsszó használata, csak pozitív számot engedünk meg:

```
LOCAL szam := 0
@ 10, 10 SAY "A szám: " GET szam VALID szam > 0
```

A következő példa azt mutatja, hogyan használjuk a WHEN által nyújtott lehetőséget, ha a GET által bekért érték egy másik GET által bekért érték függvénye. A *valtkod* (változáskód) és *ertek* mezők csak a *valtozas* mezőre adott Y válasz esetén kerülnek bekérésre. Ha a válasz N, a kurzor azonnal a *datum* mező bekérésének helyére ugrik.

```
@ 10, 10 GET Valtozas PICTURE "Y"
@ 11, 10 GET Valtkod WHEN valtozas
@ 12, 10 GET Ertek WHEN valtozas
@ 13, 10 GET Datum PICTURE "99/99/99"
READ
```

A következő példában — az előzőhöz kapcsolódva — egy másik munkaterületen megnyitott adatbázis egyik mezőjébe közvetlenül olvasunk be értéket a GET segítségével:

```
USE Sue NEW
APPEND BLANK
USE Dedi NEW
@ 1, 1 GET Sue->level
READ
@ 12, 10 GET Ertek WHEN valtozas
@ 13, 10 GET Datum PICTURE "99/99/99"
READ
```

A következő példa a @K formázó funkció használatát mutatja be. Az adatbekérés helyén megjelenik a *filenev* mező aktuális értékeként megadott PROBA1. Ha elsőként nem valamely kurzormozgató vagy a Return billentyűt nyomjuk le, akkor a mező értéke törlődik, és helyette másik írható be:

```
LOCAL allnev := "PROBA1"
@ 1, 1 SAY "Az állomány neve: " GET allnev PICTURE "@K"
READ
```

Helye: CLIPPER.LIB

Lásd még: @...SAY, READ, COL(), ROW(), SETCOLOR()

@...PROMPT

Menüpont aláfestése és üzenet definiálása

@ <sor>, <oszlop> PROMPT <menüpont> [MESSAGE <üzenet>]

<sor> és <oszlop> — numerikus kifejezések, a megjelenítésre kerülő menüpont kezdetének koordinátái a képernyőn. A sor értéke 0 és MAXROW() közé, az oszlop értéke 0 és MAXCOL() közé eshet.

<menüpont> — karakteres kifejezés, az aláfestésre kerülő menüpont.

<üzenet> — karakteres kifejezés, az aktuális menüponthoz tartozó üzenet, amely a SET MESSAGE által meghatározott sorban jelenik meg.

Minden menü maximálisan 32 menüpontot tartalmazhat. A kiválasztott menüpont a MENU TO után megadott változóban kerül eltárolásra.

A @...PROMPT parancs használata során a menüpontok közötti válogatáskor a ROW() és a COL() függvények értékei állandó aktualizáláson esnek át.

Példa:

```
LOCAL valaszt := 1
SET WRAP ON
SET MESSAGE TO 2
@ 1, 3 PROMPT "Állomány" MESSAGE "Adatállomány neve"
@ ROW(), COL() + 2 PROMPT "Szerkesztés" MESSAGE;
   "Az aktuális rekord szerkesztése"
MENU TO valaszt
```

Helye: CLIPPER.LIB

Lásd még: MENU TO, SET MESSAGE, SET WRAP, ACHOICE(), SETCOLOR()

@...SAY

Adatok megjelenítése képernyőn, illetve nyomtatón

@ <sor>, <oszlop> SAY <kifejezés> [PICTURE <formátum>]

<sor> és <oszlop> — numerikus kifejezések, a megjelenítésre kerülő szöveg kezdetének koordinátái a képernyőn, illetve nyomtatón. A sor értéke képernyő esetén 0 és MAXROW() közé, nyomtató esetén 0 és 32766 közé; az oszlop értéke képernyő esetén 0 és MAXCOL() közé, nyomtató esetén 0 és 32766 közé eshet.

SAY <kifejezés> — egy karakteres, dátum, logikai vagy numerikus kifejezés kiértékelésének eredménye, és megjelenítése az aktuális eszközön (képernyőn vagy nyomtatón).

PICTURE <formátum> — a formázó és a maszkkarakterek megjelenítése.

A @...SAY parancs hatására a <kifejezés> értéke átadásra kerül a SET DEVICE által meghatározott aktuális eszközre. Az aktuális eszköz SCREEN vagy PRINTER (képernyő vagy nyomtató) lehet. Ha az aktuális eszköz a nyomtató, akkor a @...SAY által meghatározottak nem jelennek meg a képernyőn, illetve a SET CONSOLE parancsnak nincs hatása, ha a @...SAY szerint a képernyőre történik kiíratás.

Ha az aktuális eszköz a képernyő (ez az alapértelmezés), a @...SAY parancs hatására történő kiírás után a kurzor pozíciója a kiírt szöveg utáni helyre kerül. Ezt az értéket a ROW() és a COL() függvények automatikusan felveszik. Ha a megjelenítendő kifejezés túlnyúlik a MAXROW(), illetve a MAXCOL() által meghatározott értékeken, akkor a túlnyúló rész levágásra kerül. A megjelenítés a normál színnel történik, amely a SETCOLOR() függvény segítségével állítható.

Ha az aktuális eszköz a nyomtató, a megjelenítés a megadott <sor> és <oszlop> szerinti pozícióban kezdődik. Ha a MARGIN értéke nullától eltérő, akkor ez az érték az *oszlop* értékéhez

hozzáadódik. A nyomtatófej aktuális helyzetét a PROW() és a PCOL() függvények automatikusan felveszik. Ha a <sor> kisebb az aktuális PROW() értéknél, automatikus lapdobás (EJECT — CHR(12)) következik be, majd annyi soremelés (CHR(10)), amennyi ahhoz kell, hogy a nyomtatófej a megfelelő sorba pozicionáljon. Ha az <oszlop> értéke — beleértve a SET MARGIN értéket is — kisebb az aktuális PCOL() értéknél, automatikusan soremelés (CHR(13)) következik be, majd a nyomtatófej a következő sorban a szükséges oszlopra pozicionál. Az előzőek megváltoztatása a SETPRC() függvénnyel lehetséges.

Ha az aktuális eszköz a nyomtató, akkor a SET PRINTER TO <állománynév> segítségével a kiíratás átirányítható egy lemezes állományba.

A kiíratandó szövegek a PICTURE kulcsszó után megadott módon formázhatók. A <formátum> a formázó funkciót és a maszkot tartalmazhatja. A funkciót meghatározó sztring elé a @ jelet kell írni. Egyidejüleg több funkció is megadható, felsorolásukkor nem kell közöttük üres helyet hegyni. Ha a <formátum>-ban egyidejüleg adunk meg formázó funkciót és maszkot, akkor először kötelezően a funkciónak kell állnia, amelyet a maszktól egy üres hely választ el.

A SAY és a TRANSFORM() PICTURE formázó funkciói

Funkció	Hatás
B	A számokat balra igazítja
C	A pozitív számok után CR (követel) kódot jelez
D	A dátumokat a SET DATE szerinti formában mutatja
E	A dátum típusú változót brit formában mutatja, numerikus változó esetén felcserélődik a tizedes pont és a helyiértéket jelző vessző funkciója
R	A <formátum>-ban megadott nem-funkció és nem-maszk karaktereket beszúrja a megadott helyeken
X	A DB (tartozik) megjelenítése negatív szám után (előjelet nem ír!)
Z	A vezető nullák helyett üres hely jelenik meg (zéróelnyomás)
(	A negatív számokat zárójelbe teszi (a teljes hosszt)
)	A negatív számokat zárójelbe teszi (az aktuális hosszt)
!	Az alfabetikus karaktereket nagybetűre konvertálja

A maszkok segítségével határozhatjuk meg, hogy a kiíratásra kerülő szöveg egyes karakterei milyen típusúak legyenek. Ha a maszkban a táblázatban megadott maszkkaraktereken kívül más karakter is szerepel, akkor az a megadott helyen a mező tartalmává válik (és az ottani karaktereket felülírja), kivéve, ha a @R formázó karaktert is definiáltuk.

A SAY és a TRANSFORM() PICTURE maszkkarakterei

Maszk	Hatás
A,N,X,9,#	Tetszőleges adattípus kiíratása
L	Csak a T, F, értékeket fogadja el
Y	Csak a Y és N értékeket fogadja el
!	Az alfabetikus karaktert nagybetűre konvertálja
$	A szám elején levő nullákat $-ral tölti fel
*	A szám elején levő nullákat *-gal tölti fel
.	Tizedespont megjelenítése

, A számok tagolása

Példák:

Bemutatásra kerül, hogyan lehet a PICTURE segítségével formázott kiírást végrehajtani:

```
szamla = 7125.50
bef = -125.50
tel = "062134567"
neve = "Kis B. Zsuzsa"
//
@ 1, 1 SAY szamla PICTURE "@E 9,999.99"    //"Eredmény:" 7,125.50
@ 2, 1 SAY BEF PICTURE "@)"                //"Eredmény:" (125.50)
@ 3, 1 SAY TEL PICTURE "@R (99)99/99-999"
//"Eredmény:" (06)21/34-567
@ 4, 1 SAY NEV PICTURE "@!"            //"Eredmény:" KIS B. ZSUZSA
```

A következő példában kis cimkéket állítunk elő. A SET DEVICE alkalmazásával a kiírás közvetlenül a nyomtatóra megy, a SETPRC() segítségével lapdobásokat idézünk elő.

```
USE Vevo INDEX Vevo NEW
SET DEVICE TO PRINTER
DO WHILE !EOF()                  // "Az összes rekord nyomtatása"
  @ 2, 5 SAY RTRIM(elonev) + " " + utonev
  @ 4, 5 SAY RTRIM(varos) + ", " + ker + "  " + irsz
  @ 6, 0 SAY SPACE(1)
  SETPRC(0, 0)                   // "Lapdobáshoz"
  SKIP                           // "Következö rekord"
ENDDO
SET DEVICE TO SCREEN
CLOSE Vevo
```

Helye: CLIPPER.LIB

Lásd még: @...GET, SET DEVICE, COL(), PCOL()

@...TO

Szimpla vagy duplavonalú keret rajzolása

@ <felső sor> <bal oszlop> TO <alsó sor>, <jobb oszlop> [DOUBLE]

<felső sor>, <bal oszlop>, <alsó sor>, <jobb oszlop> — numerikus kifejezések, a keret négy sarokpontja. A sarokpontok akkor látszanak a képernyőn, ha a sor 0 és MAXROW(), az oszlop 0 és MAXCOL() közötti értéket vesz fel. Ha a <felső sor> és az <alsó sor> értékei megegyeznek, egy vizszintes vonalat, ha a bal és jobb oszlop értékei egyeznek meg, függőleges vonalat húz.

DOUBLE — duplavonalú keret rajzolása. Ha nincs megadva, a keret szimpla vonalú lesz.

A keret megrajzolása után a kurzor a kereten belüli bal felső pozícióba (<felső sor>+1, <bal oszlop>+1) kerül, amely értéket természetesen a ROW() és COL() függvények is átveszik.

A @...TO annyiban különbözik a @...BOX parancstól, hogy az utóbbiban megadható egy olyan karakter is, amellyel a keret belsejét fel lehet tölteni.

Példa:

A képernyő egy részének letörlése, majd ugyanezen rész keretezése:

```
@ 10, 10 CLEAR TO 20, 40
@ 10, 10 TO 20 40 DOUBLE
```

Helye: CLIPPER.LIB

Lásd még: @...BOX, @...CLEAR

ACCEPT

Karakterlánc bevitele memóriaváltozóba

ACCEPT [<kar.kif.>] TO <változó>

<kar.kif.> — értéke a képernyőn megjelenik, majd átadásra kerül a memóriaváltozónak. Kifejezés, vagy tetszőleges adattípus lehet. Maximális hossza 255 karakter.

<változó> — a billentyűzetről beadott értéket felvevő változó. Ha még nincs definiálva, privát változóként definiálásra kerül.

Példa:

```
LOCAL cVar
ACCEPT "Bevitel: " TO cVar
IF cVar == ""
     ? "Return nyomása"
ELSE
     ? "Értéke:", cVar
ENDIF
```

Helye: CLIPPER.LIB

Lásd még: @...GET, @...SAY, INPUT, KEYBOARD, WAIT*, INKEY()

AqPPEND BLANK

Új (üres) adatrekord hozzáadása az adatbázis végéhez

APPEND BLANK

Megadása esetén ez lesz az aktuális adatbázisrekord. A mezők a "nulla" értéket veszik fel (típusuktól függően):

- karakteres: szóköz (space);
- numerikus: 0 (nulla);
- logikai: hamis (false — .F.);
- dátum: CTOD("");
- memo: üres.

Hálózatban az APPEND BLANK hatására létrejövő üres rekord lezárt (lock) állapotba kerül. Ha egy másik felhasználó FLOCK()-kal a teljes állományt, vagy LASTREC()+1 és RLOCK()-kal a rekordot kívánja zárolni, a NETERR() függvény "igaz" (.T.) értéket ad vissza. A zárolás mindaddig érvényben marad, amíg egy másik rekord zárolására, vagy az UNLOCK kiadására nem kerül sor. Az APPEND BLANK zároló hatása nem oldható fel az FLOCK() függvénnyel.

Példa:

Új rekord hozzáadása az adatbázis állományhoz, vizsgálva a NETERR() függvény értékét:

```
USE Vevo SHARED NEW
.
. <"utasítások">
.
APPEND BLANK
IF !NETERR()
   <"üres rekord létrehozása">
ELSE
   ? "Sikertelen müvelet"
   BREAK
ENDIF
```

Helye: CLIPPER.LIB.

Lásd még: APPEND FROM, FLOCK(), NETERR(), RLOCK()

APPEND FROM

Rekordok hozzáfűzése az adatbázishoz másik (.dbf vagy ASCII) adatállományból

APPEND FROM <forrás> [FIELDS <mezőlista>]
[<hatókör>] [WHILE <feltétel1>] [FOR <feltétel2>]
[SDF | DELIMITED [WITH BLANK | <elválasztó>]]

FROM <forrás> — a forrásállomány neve, amely literálként vagy karakteres kifejezésként (ekkor idézőjelek között) adható meg. A kiterjesztés elhagyása esetén az alapértelmezés: .DBF. Ha az SDF vagy a DELIMITED opciók megadásra kerültek, a kiterjesztésnek .TXT-nek kell lennie.

FIELDS <mezőlista> — a <forrás>-ból átmásolásra kerülő mezők nevei. Alapértelmezés az összes mező.

<hatókör> — a forrásállomány átmásolásra kerülő részének meghatározása:

- NEXT<n> — az első <n> rekord átmásolása;
- RECORD<n> — az <n>-ik rekord átmásolása;
- ALL — az összes rekord átmásolása (alapértelmezés).

WHILE <feltétel1> — a forrásállomány rekordjainak másolása mindaddig, amíg a feltétel nem teljesül.

FOR <feltétel2> — a forrásállomány azon rekordjainak másolása, amelyek a feltételnek megfelelnek.

SDF — ASCII állomány azonosítására. (Fix hosszúságú, „kocsivissza-soremeléssel" lezárt rekordok, az állomány végét a Ctrl-Z (1A hex) jelzi.)

DELIMITED WITH BLANK — olyan ASCII szövegállományt azonosít, ahol a mezőket a szóköz (space) választja el.

DELIMITED WITH <elválasztó> — olyan ASCII szövegállományt azonosít, ahol a mezőket az <elválasztó>-ban megadott karakter választja el. Az <elválasztó> lehet literál vagy karakteres kifejezés.

Fontos! Ha a DELIMITED WITH klauzulát alkalmazzuk, azt utolsóként kell megadni!

Az APPEND FROM parancs az aktuális adatbázishoz ASCII állományból vagy másik adatbázisból fűz rekordokat. A mezőknek azonos nevűeknek és típusúaknak kell lenniük, ellenkező esetben „runtime error" következik be.

Hálózatban az aktuális adatbázis állományt USE EXCLUSIVE módon történő megnyitással vagy az FLOCK() függvénnyel zárolni kell.

Az APPEND FROM működése alatt a <forrás> állomány „csak olvasható" (read-only) állapotba kerül.

A törlésre kijelölt rekordok változatlan formában kerülnek átmásolásra, akkor is, ha a SET DELETED ON kapcsolóállítás van érvényben.

Ha az aktuális (cél) állományban a másolásra kerülő karakteres mező hosszabb, mint a <forrás> állományban, a feltöltés üres helyekkel történik meg. Ha rövidebb, akkor a jobboldali karakterek elvesznek. Numerikus mező hosszabb adat betöltésének kisérlete hibajelzést eredményez.

Példa:

A mezőlista és a feltétel megadása:

```
USE Vevo NEW
APPEND FROM Ceglist FIELDS Ceg, Eloado, Telefon;
   FOR Ceg = 100
```

Példa a <hatókör> megadására: az 5.rekordot másoljuk át a SUE állományból:

```
APPEND RECORD 5 FROM SUE
```

Helye: CLIPPER.LIB.

Lásd még: COPY TO

AVERAGE

Numerikus kifejezések átlaga

AVERAGE <kif.lista> TO <változólista>
[<hatókör>] [WHILE <feltétel1>] [FOR <feltétel2>]

<kif.lista> — az átlagolni kívánt numerikus változók felsorolása.

<változólista> — az átlago(ka)t tartalmazó változó(k) neve(i). Ugyanannyi elemű, amennyi a <kif.lista> elemeinek száma.

<hatókör> — meghatározza az aktuális adatbázisállománynak az átlag képzésében résztvevő részét. Alapértelmezés: ALL (összes rekord).

WHILE <feltétel1> — mindaddig számolja az átlagot, amíg a feltétel igaz.

FOR <feltétel2> — azon rekordok kijelölt mezőinek átlagát számolja, amelyek a feltételnek megfelelnek.

Az AVERAGE egy vagy több numerikus kifejezés átlagát számolja ki, és helyezi el a megadott változó(k)ba. Ha a FOR feltételnek egyetlen rekord sem felel meg, 0 értéket ad vissza.

Példák:

A 100 csoportkódú cégek által vásárolt áruk átlagos értéke:

```
USE Vevo NEW
AVERAGE osszeg TO atlossz FOR ceg = "100"
```

Dátumok átlaga is kiszámítható a következő módon:

```
AVERAGE (datum - CTOD("00/00/00")) ;
   TO datumatl FOR !EMPTY(datum)
   atldatum = CTOD("00/00/00") + datumatl
```

Helye: CLIPPER.LIB.

Lásd még: SUM, TOTAL, DBEVAL()

CALL*

C vagy Assembler nyelvű eljárások hívása

CALL <eljárásnév> [WITH <paraméterlista>]

<eljárásnév> — a meghívásra kerülő külső eljárás neve.

<paraméterlista> — a meghívott eljárás számára átadásra kerülő tetszőleges adattípusú paraméter(ek) (legfeljebb hét (7) darab). Ezzel az eljáráshívással csak olyan rutinok hívhatók meg, amelyek nem adnak vissza értéket. A paraméterek a stack-be kerülnek, ahonnan a C nyelv paraméterátadási konvenciója szerint kerülnek felhasználásra. Minden paraméterhez tartozik egy FAR-pointer (négy byte), amely az aktuális paraméterértékre mutat, a WORD() függvény ezzel szemben két byte-os bináris függvényt is képes átadni. A DX:BX és az ES:BX regiszterek ugyancsak tartalmazzák a paraméterinformációk első négy byte-ját.

A CALL parancs csak a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem ajánlott. A más nyelven megírt és lefordított eljárások meghívására a Clipper 5.0 Extend System függvényeit alkalmazzuk!

A paraméterátadással szemben támasztott előírások a különböző típusoknál:

- *Karaktersztring* : A karakterargumentum FAR pointerként kerül átadásra, a két utolsó byte-nak 00 hexa értéket kell felvennie.
- *Numerikus érték*: Minden numerikus argumentum FAR pointerként kerül átadásra, mint egy nyolc byte-os IEEE lebegőpontos érték. Integer (egész) értékként való továbbításhoz használjuk a WORD() függvényt. A WORD() függvény a numerikus értéket két byte-os bináris egészre konvertálja. Megjegyzendő, hogy a WORD()

függvény nem képes 32767-nél nagyobb érték átadására, mivel két byte-on ez a maximálisan ábrázolható érték.

- *Dátum értékek*: Minden dátum argumentum FAR pointerként kerül továbbításra, mint egy négy byte-os egész, amely a nap sorszámát tartalmazza.
- *Logikai értékek*: Minden logikai argumentum FAR pointerként kerül továbbításra, mint egy két byte-os bináris egész, mely a nulla vagy az egy értéket tartalmazza az igaz (.T.), illetve a hamis (.F.) logikai értékek számára.

Követelmények a meghívott programok fordításával és szerkesztésével szemben:

- Az eljárásoknak INTEL 8086 alakú, .OBJ kiterjesztésű object file-oknak kell lenniük.
- Az eljárásoknak a C nyelv hívási és paraméterátadási konvencióit kell követni.
- Az eljárásoknak a szerkesztés során a linker számára elérhető helyen kell lenniük.

A MICROSOFT C programnyelven írt eljárásokat azon a néven kell hívni, amelyen az lefordításra került, például:

```
CALL _<függvény>
```

A dBase III PLUS programok Clipper-kompatibilissé tétele érdekében a következő utasításokat kell beírnunk .ASM állományunkba:

```
PUBLIC <függvény>
```

és

```
push ds
mov ds, dx
```

Helye: CLIPPER.LIB

Lásd még: WORD()

CANCEL*

A programfutás leállítása

CANCEL | QUIT

A CANCEL és a QUIT megállítja a futó program végrehajtását, lezárja az összes nyitott állományt és visszatér az operációs rendszer felügyelete alá. A Clipper programrendszer tetszőleges helyén kifejti hatását, ellentétben a RETURN paranccsal, amely az egy szinttel feljebb álló program számára adja át a vezérlést.

A Clipper program megszakadása során a visszatérési kódok:

- 1 — ha a folyamat fatális hibával ért véget;
- 0 — normális befejeződésnél;
- ERRORLEVEL() értéke — ha a program ezt az értéket beállította.

Helye: CLIPPER.LIB

Lásd még: QUIT, RETURN, ERRORLEVEL()

CLEAR ALL*

Alaphelyzetbe hozás

CLEAR ALL

Az összes aktív munkaterületen lezárja a nyitott állományokat (adatbázis-, index-, alternatív-, memo-), megszünteti a relációs kapcsolatokat, törli az összes *public* és *private* memóriaváltozókat, az *aktív GET*-eket, az aktuális munkaterület az 1. (SELECT 1) lesz. Ugyanakkor a CLEAR ALL **nem törli** a *local* és a *static* változókat.

A CLEAR ALL a korábbi verziókkal való kompatibilitás érdekében maradt meg, használata nem ajánlott. Az állományok lezárására használjuk a CLOSE, a memóriaváltozók törlésére a RELEASE parancsot!

Helye: CLIPPER.LIB

Lásd még: CLEAR GETS, CLEAR MEMORY, CLOSE, RELEASE

CLEAR GETS

Az aktív GET-parancsok törlése

CLEAR GETS

Törli az összes aktív GET-parancsot a GET-táblázatból, valamint megszünteti a SET KEY eljárásban, vagy a VALID klauzula utáni felhasználó által definiált függvényekben megadott READ parancsok hatását.

A GET-táblázat tartalmának törlése elérhető a CLEAR (SCREEN nélkül), valamint a READ paranccsal is.

Helye: CLIPPER.LIB

Lásd még: @...CLEAR, @...GET, CLOSE, READ, RELEASE, SET TYPEHEAD

CLEAR MEMORY

A memóriaváltozók törlése

CLEAR MEMORY

A CLEAR MEMORY törli az összes *public* és *private* memóriaváltozót. Eltérése a RELEASE ALL-tól abban áll, hogy ez utóbbi csak az adott programszinten és ez alatt definiált memóriaváltozókat törli. A *local* és *static* változókra a CLEAR MEMORY parancs nincs hatással.

Helye: CLIPPER.LIB

Lásd még: RELEASE

CLEAR SCREEN

A képernyő törlése

CLEAR [SCREEN] | CLS

A CLEAR parancs törli a képernyőt, valamint az aktív GET-eket, a kurzorpozíció a 0,0 helyzetet veszi fel. Ha a SCREEN klauzulát is megadjuk, akkor a GET-ek érvényessége megmarad. A CLS a CLEAR SCREEN kiadásával egyenértékű.

A CLEAR parancs a SET KEY és a VALID utáni eljárásokban megadott READ-re is hatással van. Ezért ha csak a képernyőt kívánjuk törölni, a CLEAR SCREEN, vagy a CLS parancsot használjuk.

Helye: CLIPPER.LIB

Lásd még: @...CLEAR, CLEAR GETS

CLEAR TYPEHEAD

A billentyűzetpuffer kiürítése

CLEAR TYPEHEAD

Kiüríti a billentyűzetpuffer tartalmát, aminek akkor vehetjük hasznát, ha nem kívánjuk megengedni a véletlenszerűen lenyomott billentyűk figyelembevételét a program futása során. Ennek a parancsnak a kiadása különösen ajánlott az ACHOICE(), illetve a DBEDIT() függvények alkalmazása előtt.

A SET TYPEHEAD és a KEYBOARD parancsok ugyancsak törlik a billentyűzetpuffer tartalmát.

Példa:

A DBEDIT() alkalmazása előtt töröljük a billentyűzetpuffer tartalmát:

```
CLEAR TYPEHEAD
DBEDIT()
```

Helye: CLIPPER.LIB

Lásd még: KEYBOARD, SET TYPEHEAD

CLOSE

Az aktuális adatbázis lezárása

CLOSE [<munkaterület>
| ALL | ALTERNATE | DATABASES | FORMAT | INDEXES]

<munkaterület> — annak a munkaterületnek a száma, ahol az *összes állományt* le akarjuk zárni.

ALL — az *összes munkaterületen* lezárja az adabázis, az alternatív és index állományokat, egyuttal törli az aktív szűrőfeltételek (FILTER) hatásait, a relációs kapcsolatokat és a format definiciókat.

ALTERNATE — a munkaterületen megnyitott alternatív állományt zárja le. Hatása azonos a SET ALTERNATE TO argumentum nélküli kiadásával.

DATABASES — lezárja az összes adatbázis, memo és kapcsolódó indexállományokat az összes munkaterületen, törli az aktív szürő és relációs feltételeket.

FORMAT — törli az aktuális formátumot. Hatása azonos a SET FORMAT TO argumentum nélküli megadásával.

INDEXES — lezárja az aktuális munkaterületen nyitott összes indexállományt.

Az opció nélkül kiadott CLOSE az aktuális adatbázist és a hozzá kapcsolódó állományokat zárja csak le, a USE paranccsal megegyező módon.

A Clipperben az állományok lezárására a következő parancsok is alkalmazhatók: QUIT, CANCEL*, RETURN (a legfelső szinten levő programban), CLEAR ALL*, USE (argumentum nélkül).

Helye: CLIPPER.LIB

Lásd még: QUIT, RETURN, SET ALTERNATE, SET INDEX, USE

COMMIT

Az aktív munkaterületek lemezre mentése

COMMIT

A COMMIT parancs hatására a Clipper belső pufferterületeinek tartalma azonnal kiíródik a lemezre, (egyébként az adatbázisok aktualizálása során csak a pufferterület megteltekor történik a tényleges lemezre írás).

E parancs segítségével kiküszöbölhetjük az esetleges áramkimaradás okozta adatvesztéseket, illetve hálózatban lehetővé tesszük más felhasználó számára az azonnali adathozzáférést.

A lemezre írás csak a 3.3, vagy ennél magasabb DOS-verziókban történik. A 3.2, és ennél alacsonyabb DOS-verziókban a COMMIT parancs a Clipper pufferterületeinek tartalmát a DOS-nak adja át.

Példa:

A következő példában az adatbázismezők új értékének meghatározása után a COMMIT paranccsal azonnal végrehajtatjuk a lemezre írást:

```
USE SUE EXCLUSIVE NEW
nev = SUE->neve
cim = SUE->cime
READ
IF UPDATED()
   APPEND BLANK
   REPLACE SUE->neve WITH nev
```

```
   REPLACE SUE->cime WITH cim
   COMMIT
ENDIF
```

Helye: CLIPPER.LIB

Lásd még: GO, SKIP

CONTINUE

A LOCATE parancs szerinti továbbkeresés

CONTINUE

A CONTINUE parancs segítségével az adatbázis aktuális rekordjától folytathatjuk az előzőleg kiadott LOCATE parancs feltétele szerinti keresést. Hatása befejeződik, ha megtalálta a feltételnek megfelelő következő rekordot, vagy az állomány végéhez ért. Ha a CONTINUE segítségével történő továbbkeresés eredményes, a megtalált rekord lesz az aktuális, a FOUND() értéke igaz (.T.) lesz, ellenkező esetben a FOUND() értéke hamis (.F.).

A különböző munkaterületeken egymástól független LOCATE feltételek lehetnek érvényben. Az egyes munkaterületeken egyszer kiadott LOCATE feltétel mindaddig érvényben marad, amíg másik LOCATE feltétellel azt felül nem írjuk. Megszüntetésére nincs külön parancs.

Az eredeti LOCATE parancs *hatókör* és WHILE feltételei a CONTINUE parancs működése során nem érvényesülnek, csak a FOR feltételt veszi figyelembe. Ha a WHILE feltételre a továbbkeresésnél is szükségünk van, használjuk a SKIP parancsot, és az eredeti LOCATE parancsot, kiegészítve a **REST** hatókörrel.

Példák:

A SZAMLA.DBF-ben keressük meg a 1014 változáskóddal rendelkező számlatulajdonosokat, majd írassuk ki ezek nevét és számlájuk követelését:

```
LOCAL kovetel := 0
USE SZAMLA NEW
LOCATE FOR SZAMLA->tulaj = "1014"
DO WHILE FOUND()
   ? SZAMLA->tulaj, kovetel += SZAMLA->egyenl
   CONTINUE
ENDDO
```

A WHILE feltétel használatának bemutatására az előző példa kibővítéseként csak azokra a számlatulajdonosokra vagyunk kíváncsiak, akik 5000 Ft feletti számlakövetelléssel rendelkeznek:

```
LOCAL kovetel := 0
USE SZAMLA INDE TULAJ NEW
SEEK "1014"
LOCATE REST WHILE SZAMLA->tulaj = "1014";
   FOR SZAMLA->egyenl > 5000
DO WHILE FOUND()
   ? SZAMLA->tulaj, kovetel += SZAMLA->egyenl
   SKIP
```

```
   LOCATE REST WHILE SZAMLA->tulaj = "1014";
     FOR SZAMLA->egyenl > 5000
ENDDO
```

Helye: CLIPPER.LIB

Lásd még: LOCATE, SEEK, EOF(), FOUND()

COPY FILE

Az állomány átmásolása egy új állományba

COPY FILE <forrás> TO <cél>

<forrás> — a másolni kívánt forrásállomány neve kiterjesztéssel együtt;

<cél> — a másolat neve kiterjesztéssel együtt.

Mindkét argumentumnál kiírhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azokat. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. A COPY FILE parancsnál a kiterjesztésre nincs alapértelmezés.

A COPY FILE parancs az aktuális (vagy a megadott elérési útvonalon levő) meghajtón és könyvtárban hajtja végre a műveletet. Ehhez a parancshoz a <forrás> állományt nem kell megnyitni (megnyitott adatbázis másolásához használjuk a COPY TO parancsot). Ha a megadott nevű <cél> állomány már létezik, azt figyelmeztetés, és hibajelzés nélkül felülírja.

Példa:

Másolatot készítünk az állományról, majd ellenőrizzük a másolat meglétét:

```
COPY FILE SUE.PRG TO DEDI.PRG
? FILE("DEDI.PRG")              // Eredmény: .T.
```

Helye: CLIPPER.LIB

Lásd még: COPY TO, RENAME, SET DEFAULT

COPY STRUCTURE

Az aktuális adatbázis-szerkezet átmásolása egy másikba.

COPY STRUCTURE [FIELDS <mezőlista>] TO <adatbázis>

FIELDS <mezőlista> — az új szerkezetbe átmásolásra kerülő mezők felsorolása. Alapértelmezés: az összes mező.

TO <adatbázis> — a létrehozásra kerülő adatbázis neve. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

A COPY STRUCTURE az aktuális adatbázis megadott mezőivel (vagy összes mezőjével) egy új, üres adatbázist hoz létre. Ha <adatbázis> nevű adatbázis már létezik, azt felülírja.

A COPY STRUCTURE előnyösen alkalmazható az adatbázis fontosabb részeinek archiválására, vagy ideiglenes adatbevitel és ellenőrzés végrehajtására.

Példák:

A példában egy ideiglenes adatbázist hozunk létre az adatbevitelhez, és az előzetes adatellenőrzést ezen hajtjuk végre:

```
vege := .T.
USE Adat NEW
COPY STRUCTURE TO Ideig
USE Ideig NEW
DO WHILE vege
   APPEND BLANK
   @ 10, 10 GET Ideig->neve
   @ 11, 11 GET Ideig->cime
   READ
   IF UPDATED()
      SELECT Adat
      APPEND BLANK
      REPLACE Adat->neve WITH Ideig->neve
      REPLACE Adat->cime WITH Ideig->cime
      SELECT Ideig
      ZAP
   ELSE
      vege := .F.
   ENDIF
ENDDO
CLOSE DATABASES
```

Helye: CLIPPER.LIB

Lásd még: COPY STRUCTURE EXTENDED, CREATE

COPY STRUCTURE EXTENDED

Adatbázis-szerkezet létrehozása adatbázismezőkből

COPY STRUCTURE EXTENDED TO <inverz adatbázis>

<inverz adatbázis> — az új (inverz) adatbázis neve. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

A COPY STRUCTURE EXTENDED egy olyan új adatbázisállományt hoz létre, amelynek tartalmát az aktuális adatbázis szerkezetét alkotó mezők alkotják, a következő szerkezetben:

Ssz	Mezőnév	Típus	Hossz	Tizedes
1	Field_name	Character	10	
2	Field_type	Character	1	
3	Field_len	Numeric	3	0
4	Field_dec	Numeric	4	0

A COPY STRUCTURE EXTENDED parancs segítségével egy futó felhasználói programmal menet közben hozható létre új adatbázis, vagy módosítható egy már létező adatbázis szerkezete. Az inverz adatbázisból a CREATE FROM paranccsal állítható elő a kívánt szerkezet. Ha egy üres inverz adatbázis-szerkezetet kívánunk csak létrehozni, akkor a CREATE parancsot használjuk.

A Clipperben a karakteres mezők megengedett maximális mérete 64K. Hogy ez a méréthatár teljesíthető legyen, a 255 byte-nál hosszabb karakteres mezőknél a COPY STRUCTURE EXTENDED parancs felhasználja a tizedesek számának definiálására szolgáló mezőt is. Ha a kilistázott adatbázisszerkezetben „teljesen szabálytalan módon" a karakteresnek definiált mezőhöz tizedes érték is tartozik, a tényleges hosszt a következőképpen számíthatjuk ki:

```
mezohossz = IF(Field_type = "C" .AND. Field_dec != 0,;
   (Field_dec * 256) + Field_len, Field_len)
```

Példa:

A fentiek szemléltetésére nézzünk egy példát, amelyből látható,hogy az inverz adatbázis tartalmát az eredeti adatbázis *szerkezete* képezi:

```
USE Vevo NEW
COPY STRUCTURE EXTENDED TO Strukt
USE Strukt NEW
LIST Field_name, Field_type, Field_len, Field_dec
```

A listázás eredménye:

```
1 CEGKOD      C  3    0
2 CEGNEV      C  20   0
3 UGYVEZ      C  20   0
4 TELEFON     C  12   0
5 FELJEGYZ    C  0    125
```

A FELJEGYZ mező tényleges hossza 125*256=32000 karakter.

Helye: CLIPPER.LIB

Lásd még: COPY STRUCTURE, CREATE, CREATE FROM, FIELD(), TYPE()

COPY TO

Az aktuális adatbázis másolása

COPY TO [FIELDS <mezőlista>] TO <cél>
[<hatókör>] [WHILE <feltétel1>] [FOR <feltétel2>]
[SDF | DELIMITED [WITH BLANK | <elválasztó>]]

FIELDS <mezőlista> — a másolásra kerülő mezők felsorolása. Alapértelmezés: az összes mező.

TO <cél> — a célállomány neve. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. Ha kiterjesztést nem adunk meg, SDF vagy DELIMITED használata esetén az alapértelmezés .txt, egyébként .dbf.

<hatókör> — az aktuális adatbázis másolásra kerülő részének meghatározása. Alapértelmezés: ALL (az összes rekord).

WHILE <feltétel1> — az aktuális adatbázis rekordjai mindaddig másolásra kerülnek, amíg a feltétel igaz.

FOR <feltétel2> — a feltételének megfelelő rekordok kerülnek csak kimásolásra.

SDF — a <cél>-állomány ASCII formátumú szövegállomány lesz. Rekordjai fix hossszúságuak, amelyeket a CR, LF „kocsivissza-soremelés" (0Dh, 0Ah) karakterek zárják le. A mezők fix hosszúságuak, határolójel nincs köztük. A karakteres mezők balra, a numerikus mezők jobbra igazítottak. A dátum mezők ÉÉÉÉHHNN formájúak, a logikai mezők a T vagy F értéket tartalmazzák. Az állomány végét az 1Ah (CHR(26)) jelzi.

DELIMITED — a <cél>-állomány az SDF által létrehozotthoz hasonló ASCII állomány, a következő sajátosságokkal: A rekordok változó hosszúságuak, amelyeket a CR, LF „kocsivisz - sza-soremelés" (0Dh, 0Ah) karakterek zárják le. A mezők változó hosszúságuak, a határolójel alapértelmezése a vessző (,). A karakteres mezők alapértelmezésben idézőjelek között szerepelnek (megváltoztatása az <elválasztó> paraméterrel lehetséges). A numerikus mezők bal oldali, illetve a karakteres mezők jobb oldali szóközzel feltöltött helyei levágásra kerülnek. A dátum mezők ÉÉÉÉHHNN formájúak, a logikai mezők a T vagy F értéket tartalmazzák. Az állomány végét az 1Ah (CHR(26)) jelzi.

DELIMITED WITH BLANK — a <cél>-állományban a mezőket egymástól egy darab szóköz karakter választja el, a karakteres mezők nincsenek idézőjelek közé téve.

DELIMITED WITH <elválasztó> — a <cél>-állományban a karakteres mezőket az <elválasztó>-ban megadott karakter határolja el egymástól. Megadhatjuk közvetlenül, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt.

Fontos! Ha a COPY vagy az APPEND parancsban a DELIMITED WITH klauzulát alkalmazzuk, azt utolsóként kell megadni!

A COPY TO parancs az aktuális adatbázist vagy annak egy részét másolja át egy másik állományba. Az új állományban a rekordok az eredeti állomány sorrendjében (nyitott indexállomány esetén ennek megfelelően) követik egymást, szűkítő feltételt a <hatókör>, a FOR|WHILE klauzula, illetve a szűrőfeltétel (FILTER) jelent. Ha a DELETED ON állapotban van, a törlésre kijelölt rekordokat nem másolja át. Ha a SET FILTER valamilyen érvényes szűrőfeltételt tartalmaz, a „nem látható" rekordok ugyancsak nem másolódnak át.

Hálózatban a <cél> állományt a Clipper EXCLUSIVE-ként nyitja meg.

Példák:

Egyszerű adatbázis másolás:

```
USE SUE NEW
COPY TO DEDI
```

Másolás ASCII állományba, határolójelek nélkül:

```
USE SUE NEW
COPY NEXT 1 TO Dedi SDF
TYPE Dedi.txt
```

Az eredmény:

```
Eset      23.3019860530T
```

Ugyanez a DELIMITED kulcsszó alkalmazásával:

```
COPY NEXT 1 TO Dedi DELIMITED
TYPE Dedi.txt
```

Az eredmény:

```
Eset,23.30,19860530,T
```

A DELIMITED WITH alkalmazásával:

```
COPY NEXT 1 TO Dedi DELIMITED WITH
TYPE Dedi.txt
```

Az eredmény:

```
'Eset',23.30,19860530,T
```

Helye: CLIPPER.LIB

Lásd még: APPEND FROM, COPY FILE, COPY STRUCTURE, SET DELETED

COUNT

A feltételnek megfelelő rekordok számlálása

COUNT TO <számláló> [<hatókör>] [WHILE <feltétel1>] [FOR <feltétel2>]

TO <számláló> — annak a változónak a neve, amelybe a számlálás eredménye kerül.

<hatókör> — az aktuális adatbázisból a számlálásra kijelölt kör. Alapértelmezés: ALL (összes rekord).

WHILE <feltétel1> — a számlálás mindaddig folyik, amíg a feltétel igaz.

FOR <feltétel2> — a feltételnek megfelelő rekordok kerülnek számlálásra.

Példák:

Számláljuk meg a 1014 kódszámú terméket vásárolt cégeket:

```
USE Vevo NEW
COUNT TO cegdb FOR kod = 1014
? cegdb
```

Az előző példában kiszámolt értéket közvetlenül egy másik adatbázis megfelelő mezőjébe írjuk:

```
USE Stat INDEX Stat
SEEK 1014
USE Vevo INDEX Vasar1
SEEK 1014
COUNT TO Stat->cegdb WHILE kod = 1014
```

Helye: CLIPPER.LIB

Lásd még: AVERAGE, SUM, TOTAL, DBEVAL()

CREATE

Üres inverz adatbázis létrehozása

CREATE <adatbázisnév>

<adatbázisnév> — az új, üres inverz adatbázis neve. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. Ha a kiterjesztést nem adjuk meg, az alapértelmezés .dbf.

A CREATE egy üres inverz adatbázis-szerkezetet alakít ki a következő formában:

Ssz	Mezőnév	Típus	Hossz	Tizedes
1	Field_name	Character	10	
2	Field_type	Character	1	
3	Field_len	Numeric	3	0
4	Field_dec	Numeric	4	0

A COPY STRUCTURE EXTENDED és a CREATE FROM parancsok együttes alkalmazásával az ilyen szerkezetű (inverz) adatbázis egyből tartalommal is feltölthető. Az <adatbázisnév> nevű adatbázis a CREATE parancs kiadásakor az aktív munkaterületen azonnal automatikusan megnyitásra kerül.

Példa:

Létrehozunk egy üres inverz adatbázist, amelyet egyetlen rekorddal töltünk fel, és ebből a CREATE FROM parancs segítségével egy egymezős adatbázis-szerkezetet hozunk létre:

```
CREATE Ideigl
APPEND BLANK
REPLACE Field_name WITH "Neve";
   Field_type WITH "C";
   Field_len WITH 25;
   Field_dec WITH 0
CLOSE
CREATE Ujdbf FROM Ideigl
```

Helye: CLIPPER.LIB

Lásd még: COPY STRUCTURE EXTENDED, CREATE FROM

CREATE FROM

Adatbázis-szerkezet létrehozása inverz adatbázisból

CREATE <új adatbázis> FROM <inverz adatbázis>

<új adatbázis> — annak az új adatbázisnak a neve, amelynek a szerkezetét az inverz adatbázis tartalmazza.

<inverz adatbázis> — az új adatbázis szerkezetét tartalmazó inverz (leíró) adatbázis neve.

Mindkét adatbázis esetében megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

A CREATE FROM parancs egy olyan új adatbázist hoz létre, amelynek szerkezetét az <inverz adatbázis> tartalmazza. Az inverz adatbázisnak a következő szerkezetűnek kell lennie:

Ssz	Mezőnév	Típus	Hossz	Tizedes
1	Field_name	Character	10	
2	Field_type	Character	1	
3	Field_len	Numeric	3	0
4	Field_dec	Numeric	4	0

Ha az inverz adatbázis a fentieknél több mezőt tartalmaz, azok a CREATE FROM parancs végrehajtása során figyelmen kivül maradnak.

Az új adatbázis a CREATE parancs hatására azonnal automatikusan megnyitódik az aktuális munkaterületen. Az <inverz adatbázis> hálózatban közös használatúként nyílik meg.

A 255 karakternél hosszabb karakteres mezők definiálása kétféleképpen lehetséges:

- a Field_len és a Field_dec mezők kombinációjával (ugyanis ha az inverz adatbázis szerkezetben „teljesen szabálytalan módon" a karakteresnek definiált mezőhöz tizedes érték is tartozik, a tényleges hosszt a Clipper következőképpen számítja ki:

```
mezohossz = IF(Field_type = "C" .AND. Field_dec != 0,;
   (Field_dec * 256) + Field_len, Field_len)
```

Ennek alapján a tényleges mezőhossz ismeretében az inverz adatbázis tartalmának kialakítása során programunkba a következő utasításokat tegyük:

```
FIELD->Field_len := <mezohossz> % 256
FIELD->Field_dec := INT(<mezohossz> / 256)
```

- A másik megoldás, ha a DBU adatbázis-karbantartó segédprogrammal (amelynek használata a szerző *„Programozási segédszoftverek I. — SAYWHAT, DBU, DEBUG"* című könyvében részletesen ismertetésre került) az inverz adatbázis szerkezetét módosítjuk olymódon, hogy a Field_len mező hosszát 5-re változtatjuk.

Példák:

Bemutatunk egy adatbázisszerkezetet interaktív módon kialakító rutint:

```
Dbletre("Ujdbf")
RETURN


FUNCTION Dbletre( ujadatb )
   CREATE Ideigl      // "üres inverz .dbf szerkezet"
   USE Ideigl
   tovabb = .T.
   DO WHILE tovabb
      APPEND BLANK
      CLEAR
      @ 5,0 SAY "Mezonev :" GET Field_name
      @ 6,0 SAY "Tipus.. :" GET Field_type
      @ 7,0 SAY "Hossz.. :" GET Field_len
```

```
      @ 8,0 SAY "Tizedes :" GET Field_dec
      READ
      tovabb := (!EMPTY(Field_name))
   ENDDO
   DELETE ALL FOR EMPTY(Field_name)
   PACK
   CLOSE
   CREATE (ujadatb) FROM Ideigl
   ERASE Ideigl.dbf
   RETURN NIL
```

Ha a fenti példában létrehozott adatbázisszerkezetet még egy 6000 karakter hosszúságú megjegyzés mezővel is ki kívánjuk egészíteni, a fenti rutint a következőkkel bővítsük ki:

```
APPEND BLANK
REPLACE Field_name WITH "megj",;
   Field_type WITH "C",;
   Field_len WITH 6000 % 256,;
   Field_dec WITH INT(6000 / 256)
```

Helye: CLIPPER.LIB

Lásd még: COPY STRUCTURE, COPY STRUCTURE EXTENDED, CREATE

DELETE

A megadott rekordok logikai törlése

DELETE [<hatókör>] [WHILE <feltétel1>] [FOR <feltétel2>]

<hatókör> — az aktuális adatbázis törlésre kijelölt része. Alapértelmezése: ha feltétel nincs megadva, az aktuális rekord, egyébként ALL.

WHILE <feltétel1> — a rekordok logikai törlése mindaddig folytatódik, amíg a megadott feltétel teljesül.

FOR <feltétel2> — azok a rekordok kerülnek logikailag törlésre, amelyekre a feltétel teljesül.

A DELETE paranccsal logikailag törölt rekordok a SET DELETED ON kapcsolóállítással csak direkt GO, illetve GOTO utasítással érhetők el, a DELETED() függvény ezekre a rekordokra vonatkozóan .T. (igaz) értéket ad vissza. Fizikai törlésük a PACK paranccsal lehetséges. A LIST és a DISPLAY parancsok ezeket a logikailag törölt rekordokat * karakterrel jelzi. A rekordok logikai törlése a RECALL paranccsal megszüntethető. Ha az összes rekordot fizikailag kívánjuk törölni, akkor a gyorsabb ZAP parancsot használjuk a DELETE ALL és PACK helyett!

Hálózatban egy rekord törlése esetén a rekordra vonatkozóan adjuk ki az RLOCK() függvényt; több rekord törlése esetén az adatbázist az FLOCK() függvénnyel kell lefoglalni, vagy már a megnyitásnál az EXCLUSIVE kulcsszót kell megadni a kizárólagos használat érdekében.

Példa:

A FOR kulcsszó alkalmazása a logikai törlésnél:

```
USE Vevo INDEX Vevo NEW
DELETE ALL FOR inaktiv
```

Helye: CLIPPER.LIB

Lásd még: PACK, RECALL, SET DELETED, ZAP, DBEVAL(), DELETED()

DELETE FILE

Állomány törlése a lemezről

DELETE FILE | ERASE <állománynév>

<állománynév> — a lemezről törölni kívánt állomány neve. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. A kiterjesztés megadása kötelező, meghajtó és elérési útvonal is megadható.

A DELETE FILE és az ERASE parancsok ugyanazt a funkciót hajtják végre. A SET DEFAULT és a SET PATH informá[illegible]k a DELETE FILE parancs végrehajtása során figyelmen kívül maradnak. Az aktuálistól eltérő meghajtón és/vagy könyvtárban való állománytörlés csak akkor hajtódik végre, ha azt az <állománynév>-ben adjuk meg.

Fontos! Az állományt törlés előtt le kell zárni. Nyitott állomány nem törölhető!

Példa:

Egy állományt törlünk a lemezről, majd ellenőrizzük a törlés végrehajtását:

```
? FILE("Nemkell.dbf")        // Eredmény: .T.
DELETE FILE Nemkell.dbf
? FILE("Nemkell.dbf")        // Eredmény: .F.
```

Helye: CLIPPER.LIB

Lásd még: CLOSE, USE, CURDIR(), FILE()

DIR*

Lemeztartalom megjelenítése

DIR [<útvonal>] [<maszk>]

<útvonal> — annak a meghajtónak és/vagy könyvtár(ak)nak a neve(i), amelyek tartalmának megjelenítését kérjük.

<maszk> — a megjeleníteni kívánt állományok maszkja, amelyben használhatók a csoportképző (*,?) karakterek is.

A paramétereket közvetlenül is megadhatjuk, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azokat. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

A DIR a megadott vagy az aktuális könyvtár tartalmát jeleníti meg a képernyőn. Ha az elérési útvonalat nem adjuk meg, akkor az aktuális könyvtár adatbázis állományaira vonatkozó információk jelennek meg: az állománynév, az utolsó módosítás időpontja, a rekordok száma. Ha elérési útvonal lett megadva, akkor a megadott könyvtár összes állományára vonatkozó információ megjelenik, az előbbinél részletesebben: állománynév, kiterjesztés, méret (byte-okban), az utolsó módosítás ideje.

Ha az elérési útvonal nincs megadva, mindig az aktuális könyvtár tartalma jelenik meg, kivéve ha a SET DEFAULT paranccsal ez felül lett bírálva.

A DIR a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem javasolt! Használjuk helyette a DIRECTORY() függvényt, amely a maszknak megfelelő állományokra vonatkozó információkat egy tömbbe helyezi, ahonnan az megjeleníthető.

Ha a könyvtárinformációkat későbbi felhasználás céljából el kívánjuk tárolni, használjuk az ADIR(), illetve az ACHOICE() függvényeket.

Példák:

Az alábbi példák a lehetséges megadási módokat mutatják:

```
maszk = *.*
DIR (maszk)          // "az összes állomány megjelenik"
DIR                  // "az összes .dbf állomány"
DIR *.prg            // "az összes .prg állomány"
```

Helye: EXTEND.LIB

Lásd még: SET DEFAULT, DIRECTORY()

DISPLAY

Rekordok megjelenítése a képernyőn

DISPLAY <kifejezéslista> [TO PRINTER] [TO FILE <állománynév>]
[<hatókör>] [WHILE <feltétel1>] [FOR <feltétel2>] [OFF]

<kifejezéslista> — mezőnevek vagy egyéb információk adhatók meg. Ezek minden rekordnál a megfelelő helyen megjelennek.

TO PRINTER — a megjelenítés a nyomtatóra történik.

TO FILE <állománynév> — a megjelenítés az <állománynév> nevű állományba kerül átirányításra. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. Ha kiterjesztést nem adunk meg, az alapértelmezés: .txt.

<hatókör> — az aktuális adatbázis megjeleníteni kívánt része. Az alapértelmezés az aktuális rekord, illetve az ezzel egyenértékű NEXT 1. Ha FOR vagy WHILE szerinti feltételt adtunk meg, akkor a <hatókör> alapértelmezése ALL.

WHILE <feltétel1> — a megjelenítés mindaddig folytatódik, amíg a feltétel teljesül.

FOR <feltétel2> — azokat a rekordokat jelenítjük meg, amelyek a feltételnek megfelelnek.

OFF — a rekordok sorszámának kiírását tíltja le.

A DISPLAY szekvenciálisan jeleníti meg az aktív munkaterületen megnyitott állomány rekordjait a <kifejezéslista> szerint. A megjelenítés tabulált formában történik, az egyes mezőket szóköz választja el egymástól. A parancs nem biztosítja fejléc kiíratását, illetve a megjelenítés bizonyos időközönkénti leállítását. A DISPLAY parancs azonos a LIST paranccsal, azzal az eltéréssel, hogy itt az alapértelmezés NEXT 1, az ottani ALL-lal szemben.

A megjelenítés alapértelmezés szerint a képernyőre történik. Ha ezt átirányítjuk a nyomtatóra vagy egy lemezállományba, akkor a képernyőn történő megjelenítés a SET CONSOLE OFF paranccsal tiltható le.

A megjelenítés megszakítása csak úgy lehetséges, ha a megszakítást a FOR kulcsszó után az INKEY() függvénnyel definiáljuk.

A nyomtatóra történő kiíratáskor a SET MARGIN szerinti beállítás érvényben lesz.

Példák:

Egy egyszerű és egy feltétel megadásával kiadott DISPLAY parancs:

```
USE Sue NEW
DISPLAY DATE(), TIME(), eset
DISPLAY eset, ido FOR ertek  25 TO PRINTER
```

A következő példa azt illusztrálja, hogyan lehet a megszakítást a DISPLAY parancsba beépíteni:

```
#define K_ESC 27
USE Sue NEW
DISPLAY eset, ido, ertek FOR INKEY() != K_ESC
```

Helye: CLIPPER.LIB

Lásd még: LIST, SET MARGIN, DBEVAL(), INKEY()

EJECT

A nyomtatófej alaphelyzetbe állítása

EJECT

Az EJECT egy lapdobás (CHR(12)) 0Ch karaktert küld a nyomtatóhoz. A PROW() és PCOL() értékeket nullázza. Ha a nyomtatás során az aktuális sorértéknél kisebb értéket adunk meg egy EJECT parancs, vagy egy SETPRC() függvény kiadása előtt, akkor a Clipper az EJECT parancsot automatikusan végrehajtja. Ezért ha lapdobás nélkül kívánjuk a nyomtatófejet az alaphelyzetbe hozni, a SETPRC() függvényt használjuk!

Példa:

Egy egyszerű nyomtatás, amelyben figyeljük a lapdobás szükségességét, és azt az EJECT parancs segítségével hajtjuk végre:

```
LOCAL sor := 99, lap := 0
USE Sue NEW
SET PRINTER ON
SET CONSOLE OFF
DO WHILE !EOF()
```

```
   IF sor > 55
      EJECT
      ? "Lap: " + LTRIM(STR(++lap, 3))
      ?? " Dátum: " + CTOD(DATE())
      ?
      ? "Esemény leírása", "Időpont"
      ?
      sor := 5
   ENDIF
   ? Sue->eset, Sue->ido
   sor++
   SKIP
ENDDO
SET PRINTER OFF
SET CONSOLE ON
CLOSE
```

Helye: CLIPPER.LIB

Lásd még: SET CONSOLE, SET DEVICE, SET PRINTER, SETPRC()

ERASE

Állomány törlése a lemezről

ERASE | DELETE FILE <állománynév>

<állománynév> — a lemezről törölni kívánt állomány neve. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. A kiterjesztés megadása kötelező, meghajtó és elérési útvonal is megadható.

Az ERASE és a DELETE FILE parancsok ugyanazt a funkciót hajtják végre. A SET DEFAULT és a SET PATH információk az ERASE parancs végrehajtása során figyelmen kívül maradnak. Az aktuálistól eltérő meghajtón és/vagy könyvtárban való állománytörlés csak akkor hajtódik végre, ha azt az <állománynév>-ben adjuk meg.

Fontos! Az állományt törlés előtt le kell zárni. Nyitott állomány nem törölhető!

Példa:

Egy állományt törlünk a lemezről, majd ellenőrizzük a törlés végrehajtását:

```
? FILE("Nemkell.dbf")        // Eredmény: .T.
ERASE Nemkell.dbf
? FILE("Nemkell.dbf")        // Eredmény: .F.
```

Helye: CLIPPER.LIB

Lásd még: CLOSE, RENAME, USE, CURDIR(), FILE(), FERASE(), FRENAME()

FIND*

Megadott kulcsérték keresése

FIND <kulcsérték>

<kulcsérték> — a keresett rekord indexkulcsa, vagy annak egy része. Karakteres kulcsoknál karaktersorozat vagy makrókifejezés, numerikus illetve dátum típusú kulcsoknál zárójelbe tett változóérték lehet. Ez utóbbi használata a SEEK paranccsal azonos végrehajtást eredményez, amivel gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

A FIND parancs hatására a rekordmutató az első olyan rekordra lép, amely a <kulcsérték>-ben megadott indexszel (indexrésszel) rendelkezik. Ha a SOFTSEEK OFF állásban van, és a keresett rekord nem létezik, a rekordmutató a LASTREC() + 1 értéket veszi fel, az EOF() *igaz* (.T.), a FOUND() *hamis* (.F.) lesz. Ha a SOFTSEEK ON állásban van, akkor a rekordmutató a keresett kulcsnál nagyobb, ahhoz legközelebb álló rekord értékét veszi fel. A FOUND() értéke ebben az esetben is *hamis* (.F.)! Az EOF() csak akkor vesz fel *igaz* (.T.) értéket, ha a keresett rekordnál nagyobb az állományban nem létezik.

A FIND a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem javasolt! Használjuk helyette a SEEK parancsot.

Példák:

A különbözőképpen megadott FIND parancsok hatása:

```
USE Sue INDEX Dedi NEW
FIND ("0530")
? FOUND(), EOF(), RECNO()      // Eredmény: .F. .T. 86
FIND 530
? FOUND(), EOF(), RECNO()      // Eredmény: .T. .F. 17
FIND "0530"
? FOUND(), EOF(), RECNO()      // Eredmény: .T. .F. 17
ido := "0530"
FIND&ido
? FOUND(), EOF(), RECNO()      // Eredmény: .T. .F. 17
ertek := 250
FIND (ertek)
? FOUND(), EOF(), RECNO()      // Eredmény: .T. .F. 23
```

Helye: CLIPPER.LIB

Lásd még: SEEK, SET ORDER, SET SOFTSEEK, EOF(), FOUND()

GO

A rekordmutató megadott rekordra helyezése

GO[TO] <rekordsorszám> | BOTTOM | TOP

<rekordsorszám> — a kívánt rekord sorszáma;

BOTTOM — az aktív munkaterület utolsó logikai rekordja;

TOP — az aktív munkaterület első logikai rekordja.

A GO az aktív munkaterületen a rekordmutatót a sorszámmal, vagy a BOTTOM, illetve TOP kulcsszóval megadott rekordra helyezi. Ha a TOP kulcsszót adtuk meg, akkor megnyitott indexállomány esetében az indexkulcs szerinti első rekordra (érvényes szűrőfeltétel esetén a szűrő szerinti első rekordra), ellenkező esetben a fizikailag első helyen levő rekordra mutat. BOTTOM esetében ugyanígy: vagy az indexkulcs szerinti utolsó rekordra, vagy a LASTREC() sorszámú rekordra mutat.

A GO <rekordsorszám> forma megegyezik a több Clipper parancsban is szereplő RECORD kulcsszó szerinti működéssel.

A GO <rekordsorszám> paranccsal a SET DELETED ON vagy a SET FILTER TO parancsokkal megadott szűrőfeltétel szerint „nem látható" rekordok is elérhetők. A GO TOP és a GO BOTTOM parancsok azonban csak a logikailag „látható" rekordokat veszi figyelembe.

A GO <rekordsorszám> paranccsal a SET UNIQUE ON, illetve az INDEX...UNIQUE parancsok következtében logikailag „nem látható" rekordok is elérhetők.

Ha a létező LASTREC() értéknél nagyobbat adunk meg <rekordsorszám>-ként, az nem vezet hibajelzéshez, hanem az EOF() és a BOF() egyaránt *igaz* (.T.), a rekordmutató pedig a LASTREC()+1 értéket veszi fel.

Hálózatban az adatbázis és az indexpuffer frissítésére alkalmazható a GO RECNO() parancs kiadása.

Példák:

A különbözőképpen megadott GO parancsok hatása:

```
USE Sue NEW
? LASTREC()              // Eredmény: 88
//
GO TOP
? RECNO()                // Eredmény: 1
//
GO BOTTOM
? RECNO()                // Eredmény: 88
//
GO 23
? RECNO()                // Eredmény: 23
//
GO 23 + 30
? RECNO()                // Eredmény: 53
```

Helye: CLIPPER.LIB

Lásd még: SET FILTER, SET RELATION, SKIP, LASTREC(), RECNO()

INDEX

Indexállomány létrehozása

INDEX ON <kifejezés> TO <indexáll> [UNIQUE]

<kifejezés> — az a kifejezés, amelynek értéke szerint a Clipper az indexállományt logikailag sorba rendezi. Karakteres, dátum, numerikus vagy logikai típusú egyaránt lehet, de a benne szereplő mezők mindegyikének azonos típusúnak kell lennie. (Ha különböznek, akkor a megfelelő függvényekkel konvertáljuk azonosra őket). Az indexkulcs maximális hossza 250 karakter lehet.

TO <indexáll> — a létrehozásra kerülő indexállomány neve. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. Ha kiterjesztést nem adunk meg, akkor az alapértelmezés szerinti lép életbe, ami Clipper kompatibilis adatbázisok esetén .ntx, dBase III Plus kompatibilis adatbázisok esetén .ndx.

UNIQUE — azt határozza meg, hogy az indexállomány egy kulcsot csak egyszer (az első előfordulásánál) tartalmaz.

Az INDEX parancs hatására létrejön egy olyan állomány, amely az aktuális adatbázis rekordjainak az <indexáll>-ban megadott indexét tartalmazza. Ha az indexállományt megnyitottuk, akkor az adatbázis rekordjai az indexkulcs szerinti sorrendben érhetők el:

- karakteres kulcsok az ASCII kódok sorrendjében;
- numerikus kulcsok növekvő számsorrendben;
- dátum típusú kulcsok kronologikus sorrendben, ahol a szóközzel feltöltött dátumok a legalacsonyabb értékként szerepelnek.;
- logikai típusú kulcsoknál az *igaz* (.T.) érték számít a nagyobb értéknek;
- memo mező nem indexelhető.

INDEX ON esetén az aktív munkaterületen levő összes nyitott indexállomány lezáródik, az új indexállomány létrehozásra kerül, egyúttal automatikusan érvénybe lép: a rekordmutató az új index szerinti első rekordra mutat.

Az INDEX parancs a SORT szerinti rendezéssel azonos módon működik, azzal a különbséggel, hogy a sorrendbe rendezett adatok fizikailag a lemezre nem íródnak fel.

Hálózatban az INDEX az indexállományt kizárólagos használatra nyitja meg.

A logikailag törölt, valamint a SET FILTER szerinti szűrőfeltétel alapján „nem látható" rekordok az indexállományban szerepelnek.

Csökkenő sorrendű indexállomány létrehozásához a DESCEND() függvény használható. Ez a függvény tetszőleges adattípust elfogad, és annak komplemensét adja vissza. Ezért, ha csökkenő indexállományban kívánunk keresni, a SEEK parancs részeként használjuk a DESCEND() függvényt!

Mivel az indexállomány sorrendje az ASCII kódok szerint kerül megállapításra, ezért szótárszerű sorrend létrehozásához az indexkulcson belül az UPPER(<kulcs>) függvény megadása szükséges.

A Clipper a dBase III PLUS kompatibilis indexállományokat egy speciális vezérlő segítségével ismeri fel. Ennek használatáról részletesebben a „*Programming and Utilities*" című dokumentációnak a *Clipper Linker*ről szóló fejezetében olvashatunk.

Fontos! Hálózatban egy dBase III PLUS kompatibilis indexállomány nem kezelhető egyidejűleg Clipper és dBase III PLUS programból! Két Clipper program azonban használhatja egyidejűleg.

Ha az indexállományt a UNIQUE kulcsszóval hozzuk létre, vagy beállítjuk a SET UNIQUE ON kapcsolót, azt érjük el, hogy az azonos kulcsértékkel rendelkező rekordok közül csak az első kerül be az indexállományba. Ha ez a kulcsérték módosul, a REINDEX vagy a PACK parancs következtében más helyre kerül, illetve fizikailag törlődik, ez csak az indexállományban szereplő első rekordot érinti.

Az indexállomány létrehozásakor ne használjuk a TRIM() függvényeket, mert az így keletkező változó hosszúságú kulcsrekordok következtében később hibás hosszúságra utaló üzenetet kaphatunk. Ha valamilyen okból mégis szükséges a használata, akkor a PAD() függvénnyel a szükséges hosszra egészítsük ki, például a következőképpen:

```
USE Sue NEW
INDEX ON PADR(RTRIM(eset) + hely, 40) TO Sueind
```

Példák:

Egyszerű indexelés egy mezővel:

```
? TYPE ("eset")              // Eredmény: C
INDEX ON eset TO Eset
```

Összetett index:

```
? TYPE ("eset")              // Eredmény: C
? TYPE ("datum")             // Eredmény: D
? TYPE ("ertek")             // Eredmény: N
INDEX ON eset + DTOS(datum) + STR(ertek)
```

Csökkenő sorrendbe indexelés:

```
USE Sue NEW
INDEX ON DESCEND(ertek) TO Ertcsok
INDEX ON DTOS(datum) + DESCEN(ertek) TO Elemez
```

Helye: CLIPPER.LIB

Lásd még: REINDEX, SET INDEX, SET ORDER, DESCEND(), DTOS()

INPUT

Egy kifejezésérték átadása memóriaváltozónak

INPUT [<szöveg>] TO <memóriaváltozó>

<szöveg> — tetszőleges típusú kifejezés, amely a képernyőn az adatbekérés helye előtt jelenik meg.

<memóriaváltozó> — az a változó, amelybe a billentyűzetrúl beadott kifejezés kerül. Ha ilyen nevű változó még nem szerepelt a programban, vagy az adott programszinten „nem látható", *privát* változóként létrehozásra kerül.

Az INPUT parancs hatására a kurzor aktuális pozíciójától egy tetszőleges típusú kifejezés (függvény vagy makró is) írható be, amely legfeljebb 255 karakter hosszú lehet. A MAXCOL() értéket túllépő adatbevitel a következő sorban folytatódik. Az adatbevitelt a RETURN lenyomásával fejezzük be. Ekkor kiértékelésre kerül a beírt kifejezés, majd értéke átadódik a <memóriaváltozó>-nak. Ha nem írtunk be semmit, az INPUT a NIL értéket adja át.

Az INPUT parancs csak a Backspace és a Return speciális billentyűk használatát engedélyezi, az ESC billentyű hatástalan. A Backspace az utolsóként beírt karaktert törli, a Return befejezi az adatbevitelt.

Példa:

Az INPUT paranccsal bevitt értéket egy lokális változónak adjuk át:

```
LOCAL szoveg
INPUT "Szövegbeírás innen: " TO szoveg
IF szoveg != NIL
   ? szoveg
ELSE
   ? "Nem írtál be szöveget"
ENDIF
```

Példa függvény beírására:

```
SET DATE ANSI
INPUT "Kérem a mai dátumot: " TO DATUM
```

A képernyőn a következőt adjuk meg:

```
DATE()
? DATUM                  // Eredmény: 91.01.17.
```

Helye: CLIPPER.LIB

Lásd még: ACCEPT

JOIN

Egy új adatbázis létrehozása két már meglévőből

JOIN WITH <másik dbf> TO <új dbf> FOR <feltétel> [FIELDS <mezőlista>]

WITH <másik dbf> — annak a munkaterületnek a neve, amelyen az aktuális munkaterületen levő adatbázissal összekapcsolni kívánt adatbázis található. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt.

TO <új dbf> — a létrehozásra kerülő adatbázis neve. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt.

Az adatbázisnevek megadásánál a zárójelbe tett karakteres kifejezés használatávalgyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

FOR <feltétel> — az új adatbázisba csak a feltételnek megfelelő rekordok kerülnek át.

FIELDS <mezőlista> — a már meglévő két adatbázis azon mezőinek neve, amelyeket a létrehozásra kerülő új adatbázisban szerepeltetni kívánunk. A nem aktuális munkaterületen levő adatbázis változóneveinek megadásánál szerepeltetni kell az adatbázis nevét (lásd a példában). Ha a FIELDS paraméter nincs megadva, akkor az aktuális munkaterület összes mezője kerül átadásra az új adatbázisnak.

A JOIN a megadott feltételeknek megfelelő rekordokból a meghatározott mezőket tartalmazó új adatbázist hoz létre.

Figyelmeztetés!

Az új adatbázis rekordjainak száma az aktuális és a másik adatbázis rekordjai számának szorzata lesz! Például, ha mindkét adatbázisunk 100 rekordot tartalmaz, az új adatbázisnál 10000 rekorddal kell számolunk. Ezért ezt a parancsot csak kivételes esetekben javasolhatjuk.

Példa:

A JOIN paranccsal egyesítjük a Sue.dbf és a Dedi.dbf tartalmát egy Suns.dbf adatbázisban:

```
USE Dedi NEW
USE Sue NEW
JOIN WITH Dedi TO Suns;
   FOR szo = Dedi->szo;
   FIELDS datum, szo Dedi->ertek, Dedi->nbog
```

Helye: CLIPPER.LIB

Lásd még: SET RELATION

KEYBOARD

A billentyűzetpuffer feltöltése

KEYBOARD <karakterlánc>

<karakterlánc> — a billentyűzetpufferbe kerülő karakterek.

A KEYBOARD parancs hatására a billentyűzetpuffer tartalma kiürítődik, majd feltöltésre kerül az itt megadott értékekkel. A következőkben a program úgy veszi, mintha az itt található értékeket a billentyűzetről kapta volna. Használata az ACCEPT, INPUT, READ, ACHOICE(), MEMOEDIT(), DBEDIT, illetve az INKEY() parancsoknál értelmezhető. A karakterlánc a 0-tól 255-ig terjedő összes INKEY() kódot tartalmazhatja, így a speciális karaktereket is! Az INKEY() kód karakteres értékre való konvertálásához a CHR() függvényt kell használni.

A KEYBOARD parancs legtipikusabb használata a SET KEY eljárással történő billentyűátdefiniálás. Ugyancsak gyakran alkalmazzák az ACHOICE() és a DBEDIT() függvények esetében.

Példa:

Az alábbi példában a KEYBOARD paranccsal egy GET szerinti értékadást valósítunk meg.

```
#include "inkey.ch"
LOCAL valt1 := SPACE(10), valt2 := 2500,;
      valt3 := SPACE(10)
```

```
CLEAR
@ 09,10 GET valt1
@ 10,10 GET valt2 WHEN Fuggv()
@ 11,10 GET valt3
READ
RETURN


FUNCTION Fuggv
   STATIC lista := { "els□", "második", "harmadik",;
          "negyedik" }
   LOCAL kep, valaszt, kulcs := LASTKEY()
   kep := SAVESCREEN(10,10,14,20)
   @ 10, 10 TO 14, 20 DOUBLE
   IF (valaszt := ACHOICE(11,11,13,19,lista)) != 0
KEYBOARD CHR(K_CTL_Y) + lista[valaszt] + CHR(kulcs)
   ENDIF
RESTSCREEN(10,10,14,20,kep)
RETURN .T.
```

Helye: CLIPPER.LIB, segédállomány az Inkey.ch

Lásd még: SET KEY, CHR(), INKEY(), LASTKEY(), NEXTKEY()

LABEL FORM

Címkék megjelenítése

LABEL FORM <címkeállomány> [TO PRINTER] [TO FILE <állománynév>] [<hatókör>] [WHILE <feltétel1>] [FOR<feltétel2>] [SAMPLE]

<címkeállomány> — a FORM definíciót tartalmazó címkeállomány (.lbl kiterjesztéssel) neve. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. Ha a kiterjesztést nem adjuk meg, az alapértelmezés .lbl.

TO PRINTER — a megjelenítés a nyomtatón történik.

TO FILE <állománynév> — a megjelenítés az <állománynév> lemezállományba irányítódik át. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. Ha a kiterjesztést nem adjuk meg, az alapértelmezés .txt.

<hatókör> — az aktuális adatbázis megjelenítendő része. Az alapértelmezés: ALL (az összes rekord).

WHILE <feltétel1> — mindaddig történik a címkék megjelenítése, amíg a feltétel hamissá (.F.) nem válik.

FOR <feltétel2> — azok a címkék jelennek meg, amelyek a megadott feltételt teljesítik.

SAMPLE — csillagokkal feltöltött tesztcímkét jelenít meg. A tesztcímke a valódi címkedefiníciónak megfelelő számú sorból és oszlopból áll. A tesztcímke megjelenítése után a „*Do you*

want more samples?" (Akar még mintát?) kérdésre adott „Y" válasz megismétli a tesztcímkét, a „N" válasz megjeleníti a feltételnek és a hatókörnek megfelelő címkéket.

A LABEL FORM az aktuális munkaterületen levő rekordok szekvenciális elérésével az .lbl kiterjesztésű címkeállományban megadott definíció alapján címzéseket (címkéket) jelenít meg. Az .lbl kiterjesztésű állomány létrehozható az RL.EXE segédprogrammal (használatát lásd a szerző „*Clipper segédprogramok*" című könyvében), vagy dBase III PLUS-szal.

A címke a képernyőn, illetve a megadott paraméterek szerint a nyomtatón jelenik meg, és/vagy átirányításra kerül egy lemezállományba. Ha a nyomtatás vagy a lemezre írás ideje alatt nem kívánjuk a képernyős megjelenítést, akkor a LABEL FORM parancs előtt a SET CONSOLE OFF kapcsolóállítást kell alkalmaznunk.

Alkalmazásakor a LABEL FORM a SET PATH-ban megadott elérési útvonalon megtalálja a kívánt címkeállományt, ha az az aktuális könyvtárban nem szerepel.

Hálózatban a LABEL FORM közös használatra nyitja meg az állományt.

A LABEL FORM parancs menetközbeni megszakításához az INKEY() függvényt kell beépítenünk a FOR feltételei közé (lásd a későbbi példát).

A nyomtatón történő megjelenítés esetén a LABEL FORM figyelembe veszi az aktuális SET MARGIN értéket.

Példák:

A címkék megjelenítése átirányításra kerül egy lemezállományba. Két lehetséges megadási formát mutatunk be:

```
LOCAL cimke := "Vevok", adatall := "Vevok"
USE Vevok INDEX Vevok NEW
LABEL FORM Vevok TO PRINTER TO FILE Vevok
LABEL FORM (cimke) TO PRINTER TO FILE (adatall)
```

A példa azt mutatja be, hogyan lehet az INKEY() függvény beépítésével a LABEL FORM működése közbeni megszakítást biztosítani:

```
#define K_ESC  27
USE Vevok INDEX Vevok NEW
LABEL FORM Vevok FOR INKEY() != K_ESC
```

Helye: CLIPPER.LIB

Lásd még: REPORT FORM, SET PRINTER

LIST

Rekordok megjelenítése

LIST <kifejezéslista> [TO PRINTER] [TO FILE <állománynév>]
[<hatókör>] [WHILE <feltétel1>] [FOR<feltétel2>] [OFF]

<kifejezéslista> — mezőnevek vagy egyéb információk adhatók meg. Ezek minden rekordnál a megfelelő helyen megjelennek.

TO PRINTER — a megjelenítés a nyomtatóra történik.

TO FILE <állománynév> — a megjelenítés az <állománynév> nevü állományba kerül átirányításra. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. Ha kiterjesztést nem adunk meg, az alapértelmezés: .txt.

<hatókör> — az aktuális adatbázis megjeleníteni kívánt része. Az alapértelmezés: ALL.

WHILE <feltétel1> — a megjelenítés mindaddig folytatódik, amíg a feltétel teljesül.

FOR <feltétel2> — azokat a rekordokat jelenítjük meg, amelyek a feltételnek megfelelnek.

OFF — a rekordok sorszámának kiírását tíltja le.

A LIST szekvenciálisan jeleníti meg az aktív munkaterületen megnyitott állomány rekordjait a <kifejezéslista> szerint. A megjelenítés tabulált formában történik, az egyes mezőket szóköz választja el egymástól. A parancs nem biztosítja fejléc kiíratását, illetve a megjelenítés bizonyos időközönkénti leállítását. A LIST parancs azonos a DISPLAY paranccsal, azzal az eltéréssel, hogy itt az alapértelmezés ALL, az ottani NEXT 1-gyel szemben.

A megjelenítés alapértelmezés szerint a képernyőre történik. Ha ezt átirányítjuk a nyomtatóra vagy egy lemezállományba, akkor a képernyőn történő megjelenítés a SET CONSOLE OFF paranccsal tíltható le.

A megjelenítés megszakítása csak úgy lehetséges, ha a megszakítást a FOR kulcsszó után az INKEY() FÜGGVÉNNYEL definiáljuk.

A nyomtatóra történő kiíratáskor a SET MARGIN szerinti beállítás érvényben lesz.

Példák:

Egy egyszerű és egy feltétel megadásával kiadott LIST parancs:

```
USE Sue NEW
LIST DATE(), TIME(), eset
LIST  eset, ido FOR ertek > 25 TO PRINTER
```

A következő példa azt illusztrálja, hogyan lehet a megszakítást a LIST parancsba beépíteni:

```
#define K_ESC   27
USE Sue NEW
LIST eset, ido, ertek FOR INKEY() != K_ESC
```

Helye: CLIPPER.LIB

Lásd még: ?|??, DISPLAY

LOCATE

Feltételnek megfelelő rekord keresése

LOCATE [<hatókör>] FOR <feltétel1> [WHILE <feltétel2>]

<hatókör> — az aktuális adatbázis azon része, amelyen a LOCATE szerinti keresést végezni kívánjuk.

FOR <feltétel1> — a következő keresendő rekord meghatározása. A feltétel a hatókörön belül érvényes.

WHILE <feltétel2> — annak a meghatározása, hogy a keresés milyen feltétel teljesülése után álljon le. Ha a <feltétel2>-ben megadott feltétel hamissá (.F.) válik, a keresés nem folytatódik.

A LOCATE parancs segítségével az aktuális munkaterületen megnyitott adatbázis megadott hatókörében kereshetjük meg a kívánt feltételeknek megfelelő első rekordot. A LOCATE első alkalommal történő kiadásakor a keresés a hatókörnek megfelelő első rekordtól kezdődik, és mindaddig tart, amíg a feltételnek megfelelő első rekordot meg nem találja, vagy ilyen rekordot egyáltalán nem talál és a hatókör végére ér. Ha talált rekordot, a FOUND() függvény *igaz* (.T.), ha nem talált, *hamis* (.F.).

A különböző munkaterületeken egymástól független LOCATE feltételek lehetnek érvényben. Az egyes munkaterületeken egyszer kiadott LOCATE feltétel mindaddig érvényben marad, amíg másik LOCATE feltétellel azt felül nem írjuk. Megszüntetésére nincs külön parancs.

A LOCATE parancs a CONTINUE paranccsal együtt alkalmazható. Ha az első LOCATE parancs kiadása után a keresést az aktuális rekordtól kívánjuk folytatni, akkor a CONTINUE parancsot kell kiadni. A LOCATE parancs ismételt kiadása minden esetben az állomány (hatókör) elejétől kezdve kezdi a keresést.

A LOCATE parancs *hatókör* és WHILE feltételei a CONTINUE parancs működése során nem érvényesülnek, csak a FOR feltételt veszi figyelembe. Ha a WHILE feltételre a továbbkeresésnél is szükségünk van, a CONTINUE helyett használjuk a SKIP parancsot, és az eredeti LOCATE parancsot, kiegészítve a **REST** hatókörrel (LOCATE REST WHILE <feltétel>).

A LOCATE szerinti keresés index nélküli állományokon, és tetszőleges mezőkre megadott feltételeket figyelve is végrehajtható, ellentétben a FIND és a SEEK parancsokkal.

Példák:

Tipikus LOCATE parancsok:

```
USE Szamla INDEX Kod
LOCATE FOR tulaj = "1014"
? FOUND(), EOF(), RECNO()      //"Eredmény:" .T. .F. 5
LOCATE FOR tulaj = "1208"
? FOUND(), EOF(), RECNO()      //"Eredmény:" .F. .T. 86
```

A SZAMLA.DBF-ben keressük meg a 1014 változáskóddal rendelkező számlatulajdonosokat, akik 5000 Ft-nál nagyobb számlakövetelléssel rendelkeznek, majd írassuk ki ezek nevét és számlájuk követelését:

```
SEEK "1014"
LOCATE FOR egyenl > "5000" WHILE tulaj = "1014"
DO WHILE FOUND()
   ? tulaj, egyenl
   SKIP
   LOCATE REST FOR egyenl > 5000 WHILE tulaj = "1014";
ENDDO
```

Helye: CLIPPER.LIB

Lásd még: CONTINUE, SEEK, SET FILTER, EOF(), FOUND()

MENU TO

Menüből történő választás végrehajtása

MENU TO <memóriaváltozó>

<memóriaváltozó> — annak a mamóriaváltozónak a neve, amely a kiválasztott menüpontnak megfelelő sorszám értékét veszi át. Ha ez még nincs specifikálva, vagy a programszerkezet adott helyéről „nem látható", akkor privát változóként létrehozásra kerül.

A MENU TO parancs a Clipper menürendszerének kiválasztási mechanizmusa. A MENU TO parancs előtt meg kell határozni magát a menürendszert (a @...PROMPT parancsokkal), illetve szükség esetén az egyes menüpontokhoz kapcsolódó megjegyzéseket a MESSAGE kulcsszó után. Ezután történhet a menü aktivizálása a MENU TO <memóriaváltozó> paranccsal. Ha a <memóriaváltozó> még nincs definiálva, az első, egyébként a memóriaváltozó aktuális értékének megfelelő menüpont világít.

Az egyes menüpontok az aktuális szabványszínnel jelennek meg, a megvilágított (kiválasztás alatt álló) menüpont pedig az aktuális kiemelőszínnel látható.

A kurzormozgató nyílakkal mozoghatunk az előző vagy a következő menüpontra. Amelyik menüponton állunk, az ahhoz tartozó megjegyzés (amit a MESSAGE kulcsszó után adtunk meg) a SET MESSAGE szerint meghatározott sorban megjelenik. Ha a WRAP kapcsoló ON állásban van, az első menüponton állva a felfelé nyílat nyomva az utolsó menüpontra, illetve fordítva: az utolsó menüponton állva és a lefelé mutató nyílat nyomva az első menüpontra léphetünk.

A szükséges menüpont kiválasztása a RETURN, a PgUp, PgDn billentyű, vagy a menüpont első betűjének lenyomásával történik. Ekkor a MENU TO parancs a kiválasztott menüpont sorszámát numerikus értékként átadja a megadott memóriaváltozónak. Az ESC billentyű lenyomása esetén a menükiválasztás befejeződik, a memóriaváltozónak visszaadott érték nulla (0). A MENU TO érvényességi ideje alatt aktív funkcióbillentyűket, és azok hatását a táblázat mutatja.

A SET KEY eljárások alkalmazhatók a MENU TO parancs működése idején, méghozzá úgy, hogy a függőben levő PROMPT-ok nem törlődnek.

Kulcs	Működés
felfelé nyíl	az előző menüpont
lefelé nyíl	a következő menüpont
Home	az első menüpont
End	az utolsó menüpont
balra mutató nyíl	az előző menüpont
jobbra mutató nyíl	a következő menüpont
PgUp	a menüpont kiválasztása
PgDn	a menüpont kiválasztása
Return	a menüpont kiválasztása
ESC	a kiválasztás megszakítása, nullát ad vissza
első betű	a megadott betűvel kezdődő első menüpont kiválasztása

Példa:

A példa egy egyszerű függőleges beállítású menürendszerre példa, ahol a megjegyzések, üzenetek a 23. sor közepén jelennek meg. A menü megjelenésekor a 2. menüpont lesz a megvilágított:

```
LOCAL valaszt := 2
SET WRAP ON
SET MESSAGE TO 23 CENTER
@ 6, 10 PROMPT "Íj felvitel" MESSAGE "Íj kód felvitele"
@ 7, 10 PROMPT "Módosítás" MESSAGE "Mezömódosítás"
@ 8, 10 PROMPT "Törlés" MESSAGE "A kívánt kód törlése"
@ 10, 10 PROMPT "Kilépés" MESSAGE "Vissza a DOS-ba"
MENU TO valaszt
//
DO CASE
   CASE valaszt = 0 .OR. valaszt = 4
      QUIT
   CASE valaszt = 1
      Ujkod()
   CASE valaszt = 2
      Modosit()
   CASE valaszt = 3
      Torol()
ENDCASE
RETURN
```

Helye: CLIPPER.LIB

Lásd még: @...PROMPT, SET MESSAGE, SET WRAP, ACHOICE()

NOTE*

Megjegyzéssor elhelyezése a programban

NOTE [<megjegyzéssor>]

<megjegyzéssor> — a megjegyzést képző karaktersorozat.

A NOTE a * szinonímája. A NOTE kulcsszó és a sorvég (CR, LF) között beírt karaktereket a Clipper fordító figyelmen kívül hagyja. A megjegyzés a következő sorban nem folytatható.

Ha többsoros megjegyzést kívánunk a programba írni, a megjegyzésblokk elejét a /*, végét a */ karakterekkel jelöljük.

A NOTE a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem javasolt! Használjuk helyette a C programozási nyelv stílusában a /* */, vagy a // jeleket. Ugyancsak használható helyette a dBase stílusában a *, valamint a &&.

Példa:

A Clipper 5.0 különböző kommentármegadási lehetőségei:

```
// "Ez egy megjegyzés"
/* "Ez egy megjegyzés" */
/* "Ez egy kétsoros
megjegyzés" */
* "Ez egy megjegyzés"
&& "Ez egy megjegyzés"
NOTE "Ez egy megjegyzés"
```

PACK

Törlésre kijelölt rekordok fizikai eltávolítása

PACK

A PACK parancs fizikailag is eltávolítja az aktuális adatbázis állományból az előzőleg logikailag már törölt (törlésre kijelölt) rekordokat. Az aktuális munkaterületen megnyitott összes indexállományban újraindexelést hajt végre (REINDEX), a törölt rekordok helyét a lemezen szóköz karakterekkel tölti fel. A PACK parancs nem képes az adatbázis előző állapotát tartalmazó „backup" állományt készíteni. A PACK parancs végrehajtása után a rekordmutató az aktuális munkaterület logikailag első rekordjára mutat.

Hálózatban a PACK parancs előtt az állományt zárolni kell. Ez vagy már a megnyitásnál (USE <adatbázis> EXCLUSIVE), vagy később az FLOCK() függvénnyel történhet. Az állomány zárolásának elmaradása hibajelzéshez vezet.

Nagy adatbázis esetében a PACK parancs elég időigényes művelet, ezért — különösen hálózatban — csak a legszükségesebb esetekben alkalmazzuk!

Példa:

Egy egyszerű PACK hatása:

```
USE Sue NEW
? LASTREC()          // Eredmény: 91
//
DELETE RECORD 17
PACK
? LASTREC()          // Eredmény: 90
```

Helye: CLIPPER.LIB

Lásd még: DELETE, RECALL, REINDEX, ZAP, DELETED(), FLOCK()

QUIT

A programrendszer futásának leállítása

QUIT | CANCEL*

A QUIT és a CANCEL megállítja a futó program végrehajtását, lezárja az összes nyitott állományt és visszatér az operációs rendszer felügyelete alá. A Clipper programrendszer tetszőleges helyén kifejti hatását, ellentétben a RETURN paranccsal, amely az egy szinttel feljebb álló program számára adja át a vezérlést.

A Clipper program megszakadása során a visszatérési kódok:

- 1 — ha a folyamat fatális hibával ért véget;
- 0 — normális befejeződésnél;
- ERRORLEVEL() értéke — ha a program ezt az értéket állította.

Példa:

Példa arra, hogyan lehet a párbeszéd céljára szolgáló ablak információja alapján végrehajtani a program futásának leállítását:

```
IF parbesz(10, 10, "Kilépés a DOS-ba", "BG+/B,B/W", 2)
   QUIT
ENDIF
RETURN
```

Helye: CLIPPER.LIB

Lásd még: BEGIN SEQUENCE, RETURN, ERRORLEVEL()

READ

A GET-ek szerinti adatbekérések aktíviziálása

READ [SAVE]

SAVE — megadása esetén a READ parancs végrehajtása után is megőrződnek a GET-lista értékei, így egy következő READ-del újra aktívizálhatók. Ha nincs megadva, a READ után a GET-lista elemei törlődnek.

A READ teljes képernyős („full-screen") üzemmódban olvassa be az előző READ óta beírt, módosított, javított GET-lista elemeket. A képernyőn levő GET-lista elemei mindaddig módosíthatók, amíg a CLEAR, CLEAR GETS, CLEAR ALL vagy a READ parancs kiadásra nem kerül.

A GET-ek közül a megadott WHEN <feltétel>-t nem teljesítők figyelmen kívül maradnak. Ha a GET parancssorban RANGE vagy VALID ellenőrzőértékek szerepelnek, akkor először kiértékelésre kerül, hogy a GET után beadott érték teljesíti-e az ellenőrzőértékek által előírt feltételeket. Ha az eredmény *igaz* (.T.), akkor sor kerülhet a következő GET-érték beolvasására, egyébként az aktuális GET-érték megadását mindaddig meg kell ismételni, amíg az előírt feltételeknek megfelelő választ nem sikerül megadni, vagy az ESC billentyűt le nem nyomjuk.

A READ parancs navigációs billentyűi:

Billentyű	Hatás
balra nyíl, Ctrl-S	egy karakterpozíció balra, de nem lép át az előző GET-mezőbe
jobbra nyíl, Ctrl-D	egy karakterpozíciót jobbra lép, de nem lép át a következő GET mezőbe
Ctrl-balra nyíl, Ctrl-A	az előző szó elejére áll
Ctrl-jobbra nyíl, Ctrl-F	a következő szó elejére áll
felfelé nyíl, Ctrl-E	az előző GET-mező elejére áll
lefelé nyíl,Ctrl-X,Ctrl-M,Return	a következő GET-mező elejére áll
Home	az aktuális GET-mező elejére áll
End	az aktuális GET-mező végére áll
Ctrl-Home	a legelső GET-mező elejére áll
Ctrl-End	az legutolsó GET-mező elejére áll

A READ parancs szerkesztő billentyűi

Billentyű	Hatás
Del, Ctrl-G	a kurzor alatti karakter törlése
Backspace, Ctrl-H	a kurzor előtti karakter törlése
Ctrl-T	a jobbra levő szó törlése
Ctrl-Y	a kurzortól a mező végéig töröl
Ctrl-U	visszaállítja a mező eredeti tartalmát
Ins, Ctrl-V	átváltás beszúrás üzemmódba

A READ parancs befejező billentyűi

Billentyű	Hatás
Ctrl-W,Ctrl-C,PgUp,PgDn	az aktuális GET szerkesztésének befejezése az értékek elmentésével
Return, Ctrl-M	az előző GET szerkesztésének befejezése
Esc	az aktuális GET szerkesztésének befejezése az érték elmentése nélkül
felfelé nyíl	az első GET-mezőnél kiadva: a szerkesztés befejezése, ha a READEXIT()=.T.
lefelé nyíl	az utolsó GET-mezőnél kiadva: a szerkesztés befejezése, ha a READEXIT()=.T.

A SET KEY, a VALID, a WHEN eljárásokkal, vagy a felhasználó által definiált függvényekkel egymásba ágyazott READ-ek hajthatók végre úgy, hogy új GET-listát készítünk a @...GET parancsokkal, majd kiadjuk a READ parancsot. A függvény (eljárás) befejeződése után az új GET-lista törlődik, és az eredeti GET-lista ismét érvénybe lép (lásd példa).

A READ parancs hatása megszűnik a BREAK, CLEAR, CLEAR GETS, CLEAR ALL parancsok hatására.

Ha az aktuális GET-lista valamelyik mezőjének értéke módosítva lett, akkor az UPDATED() függvény *igaz* (.T.) értéket vesz fel.

Példák:

Egyszerű értékellenőrzéses GET-ek:

```
CLEAR
valt1 := valt2 := valt3 :=SPACE(10)
@ 10, 10 SAY "Egy: " GET valt1 VALID !EMPTY(valt1)
@ 11, 10 SAY "Kettö: " GET valt2 WHEN RTRIM(valt2) != "Sue"
@ 12, 10 SAY "Három: " GET valt3 VALID !EMPTY(valt3)
READ
```

A következő példa azt mutatja be, hogyan lehet a SET KEY, a VALID, a WHEN eljárásokkal, vagy felhasználó által definiált függvénnyel egymásba ágyazott READ-eket létrehozni:

```
LOCAL nev := SPACE(10)
@ 10, 10 GET nev VALID Formell( nev )
READ
RETURN
FUNCTION Formell( vizsgal )
   LOCAL Getlista := {}      // "új GET-lista létrehozása"
   USE Vevok INDEX Vevok NEW
   SEEK vizsgal
   IF FOUND()
      @ 15, 10 GET ugyint     // "az új GET-listába"
      @ 16, 10 GET telefon
      READ                 // "az új GET-lista szerint"
   ENDIF
   CLOSE Vevok
   RETURN .T.              // "az új GET-lista törlése"
```

Helye: CLIPPER.LIB

Lásd még: @...GET, CLEAR GETS, LASTKEY(), READEXIT(), READMODAL()

RECALL

A rekordok logikai törlésének megszüntetése

RECALL [<hatókör>] [FOR <feltétel1>] [WHILE <feltétel2>]

<hatókör> — az aktuális adatbázis meghatározott részének kijelölése. Alapértelmezés: az aktuális rekord, azaz NEXT 1. Ha a feltételek valamelyike is meg lett adva, akkor az alapértelmezés: ALL.

WHILE <feltétel2> — a logikailag törölt rekordok visszaállítása mindaddig folyik, amíg a feltétel igaz.

FOR <feltétel1> — azoknak a logikailag törölt rekordoknak a visszaállítása, amelyekre a feltétel teljesül (a hatókörön belül érvényes).

A RECALL az aktuális munkaterületen a törlésre kijelölt rekordok logikai törlését szünteti meg. A DELETE parancs inverze. A RECALL DELETED ON kapcsolóállásban kifejti hatá-

sát. Ha a logikai törlést a PACK parancs kiadása követte, akkor a törölt rekordok visszaállítása már nem lehetséges!

Hálózatban az aktuális rekord visszaállításához az RLOCK(), több rekord visszaállításához az FLOCK() függvénnyel történő zárolás szükséges, vagy már az adatbázist kizárólagos használatra kell megnyitni (USE ... EXCLUSIVE).

Példa:

Példa a RECALL parancs hatására:

```
USE Sue NEW
//
DELETE RECORD 88
? DELETED()              // Eredmény: .T.
//
RECALL
? DELETED()              // Eredmény: .F.
```

Helye: CLIPPER.LIB

Lásd még: DELETE, PACK, SET DELETED, ZAP, DELETED(), FLOCK(), RLOCK()

REINDEX

Az aktuális munkaterület nyitott indexállományainak újraépítése

REINDEX

A REINDEX parancs az aktuális munkaterület összes nyitott indexállományát újraépíti. Az újraindexelés befejezésekor minden eddig is nyitott indexállomány nyitva lesz, a SET ORDER értéke 1-re áll, a rekordmutató az aktuális index szerint logikailag első rekordra mutat. Ha valamely indexállomány SET UNIQUE ON-nal lett létrehozva, a REINDEX csak az egyedi indexeket veszi figyelembe.

Hálózatban az aktuális adatbázist kizárólagos használatra kell megnyitni (USE ... EXCLUSIVE).

Fontos!

A REINDEX nem működik helyesen, ha a kulcsmezőn valamilyen változtatás történt, például növekedett a hossza. Ilyenkor az INDEX ON paranccsal újra kell indexelni az állományt. Ha biztosak akarunk lenni abban, hogy indexállományunk helyes, akkor minden esetben inkább az INDEX ON parancsot használjuk, és mellőzzük a REINDEX alkalmazását!

Példa:

Példa a REINDEX kulcsszó elhelyezésére

```
USE Sue INDEX Sue, Dedi NEW
REINDEX
```

Helye: CLIPPER.LIB

Lásd még: INDEX, PACK, SET INDEX, USE

RELEASE

A „public" és „private" memóriaváltozók törlése

RELEASE <memóriaváltozó-lista>
RELEASE ALL [LIKE | EXCEPT <maszk>]

<memóriaváltozó-lista> — a törlésre kijelölt *private* és *public* változók, illetve tömbök listája.

ALL — az összes memóriaváltozó törlésre kerül.

LIKE <maszk> — a maszk által meghatározott változók kerülnek törlésre.

EXCEPT <maszk> — a maszk által meghatározott változók **nem** kerülnek törlésre.

A <maszk> tartalmazhatja a csoportképző (?, *) karaktereket is.

A RELEASE a megadott két mód valamelyike szerint működik. Ha a <memóriaváltozó-lista>-val együtt adjuk meg, akkor a meghatározott *private* és *public* változók, valamint tömbök törlődnek a memóriából. A törlés csak az aktuális programra, illetve a programrendszerben alatta elhelyezkedő programokra érvényes.

Ha a RELEASE parancsot az ALL kulcsszó utáni valamely lehetőséggel adtuk ki, a privát memóriaváltozók az aktuális programban a NIL értéket veszik fel, és nem törlődnek mindaddig, amíg az aktuális eljárás, vagy felhasználó által definiált függvény végrehajtása be nem fejeződik. A *public* változók a RELEASE parancsnak ebben a formában történő kiadásánál érintetlenek maradnak. Ezek törlése csak a RELEASE másik formájában való kiadásával, vagy a CLEAR MEMORY paranccsal lehetséges.

A *local* és a *static* változókra a RELEASE parancs nincs hatással. A *local* változók automatikusan törlődnek az eljárás vagy a felhasználó által definiált függvény befejeződése után. A *static* változók nem törölhetők.

Helye: CLIPPER.LIB

Lásd még: CLEAR MEMORY, LOCAL, PRIVATE, PUBLIC, QUIT

RENAME

Az állomány nevének megváltoztatása

RENAME <régi név> TO <új név>

<régi név> — annak az állománynak a neve és kiterjesztése, (szükség esetén a meghajtóval és az elérési útvonallal együtt), amelynek az elnevezését meg kívánjuk változtatni. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

<új név> — az állománynak a változtatás utáni új neve és kiterjesztése. Szükség esetén meg kell adni a meghajtót és az elérési útvonalat is. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

A RENAME segítségével megváltoztathatjuk egy állománynak a nevét. Ha a régi névben megadott meghajtó és/vagy elérési útvonal eltér az új névben megadottól, akkor az állomány au-

RELEASE

A „public" és „private" memóriaváltozók törlése

RELEASE <memóriaváltozó-lista>
RELEASE ALL [LIKE | EXCEPT <maszk>]

<memóriaváltozó-lista> — a törlésre kijelölt *private* és *public* változók, illetve tömbök listája.

ALL — az összes memóriaváltozó törlésre kerül.

LIKE <maszk> — a maszk által meghatározott változók kerülnek törlésre.

EXCEPT <maszk> — a maszk által meghatározott változók nem kerülnek törlésre.

A <maszk> tartalmazhatja a csoportképző (?, *) karaktereket is.

A RELEASE a megadott két mód valamelyike szerint működik. Ha a <memóriaváltozó-lista>-val együtt adjuk meg, akkor a meghatározott *private* és *public* változók, valamint tömbök törlődnek a memóriából. A törlés csak az aktuális programra, illetve a programrendszerben alatta elhelyezkedő programokra érvényes.

Ha a RELEASE parancsot az ALL kulcsszó utáni valamely lehetőséggel adtuk ki, a privát memóriaváltozók az aktuális programban a NIL értéket veszik fel, és nem törlődnek mindaddig, amíg az aktuális eljárás, vagy felhasználó által definiált függvény végrehajtása be nem fejeződik. A *public* változók a RELEASE parancsnak ebben a formában történő kiadásánál érintetlenek maradnak. Ezek törlése csak a RELEASE másik formájában való kiadásával, vagy a CLEAR MEMORY paranccsal lehetséges.

A *local* és a *static* változókra a RELEASE parancs nincs hatással. A *local* változók automatikusan törlődnek az eljárás vagy a felhasználó által definiált függvény befejeződése után. A *static* változók nem törölhetők.

Helye: CLIPPER.LIB

Lásd még: CLEAR MEMORY, LOCAL, PRIVATE, PUBLIC, QUIT

RENAME

Az állomány nevének megváltoztatása

RENAME <régi név> TO <új név>

<régi név> — annak az állománynak a neve és kiterjesztése, (szükség esetén a meghajtóval és az elérési útvonallal együtt), amelynek az elnevezését meg kívánjuk változtatni. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

<új név> — az állománynak a változtatás utáni új neve és kiterjesztése. Szükség esetén meg kell adni a meghajtót és az elérési útvonalat is. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

A RENAME segítségével megváltoztathatjuk egy állománynak a nevét. Ha a régi névben megadott meghajtó és/vagy elérési útvonal eltér az új névben megadottól, akkor az állomány au-

tomatikusan áthelyezésre kerül az új könyvtárba. Az állomány megtalálásához a RENAME nem tudja értelmezni a SET DEFAULT és a SET PATH beállításokat, így az átnevezés csak akkor hajtódik végre, ha a <régi név> állomány az aktuális könyvtárban, vagy a megadott elérési útvonalon található. Ha <új név> néven már létezik állomány, vagy a <régi név> állomány nyitva van, az átnevezés nem hajtódik végre. Ezért ezt az átnevezés előtt a FILE() függvénnyel ellenőrízzük!

Fontos! Átnevezés előtt az állományt le kell zárni! Nyitott állomány átnevezése beláthatatlan következményekkel járhat! Ne felejtsük el, hogy a .dbf állománnyal együtt az esetleges memo (.dbt) állományt is át kell nevezni!

Példa:

Példa az átnevezésre:

```
regi := "regiall.txt"
ujal := "ujall.txt"
IF !FILE(ujal)
   RENAME (regi) TO (ujal)
ELSE
   ? "Már létezö állomány"
ENDIF
```

Helye: CLIPPER.LIB

Lásd még: COPY FILE, ERASE, RUN, CURDIR(), FILE(), FERASE(), FRENAME()

REPLACE

Értékadás az adatbázismezők számára

REPLACE <mezőnév1> WITH <kifejezés1>
[,<mezőnév2> WITH <kifejezés2>...]
[<hatókör>] [WHILE <feltétel1>] [FOR <feltétel2>]

<mezőnévx> — annak a mezőnek a neve, amelynek új értéket kívánunk adni. Tetszőlegesen lehet numerikus, karakteres, dátum, logikai, vagy memo változó. A megadott mezőnevet tartalmazó adatbázisnak nyitva kell lennie!

<kifejezésx> — a <mezőnévx>-nek adott új érték. Lehet változónév, vagy „direkt" érték.

<hatókör> — az aktuális adatbázis azon része, amelyre az értékadás vonatkozik. Az alapértelmezés az aktuális rekord, azaz NEXT 1. Ha az adatbázis összes rekordjában (vagy a WHILE, illetve a FOR feltételekkel megadott részében) azonos értéket kívánunk a mezőnek adni, az ALL hatókört használjuk.

WHILE <feltétel1> — az adott mező(k) számára a <kifejezés> szerinti érték kerül átadásra az aktuális rekordtól kezdve mindaddig, amíg a feltétel igaz.

FOR <feltétel2> — az adott mező(k) számára a feltételnek megfelelő rekordokban kerül a <kifejezés> szerinti érték átadásra.

A REPLACE paranccsal új értéket adhatunk egy megnyitott adatbázis aktuális rekordja egy vagy több mezőjének. Ugyanazt hajtja végre, mint a := operátor.Ha a := értékadást kívánjuk

használni, az alias név megadása kötelező!. Az aktuális munkaterületen levő adatbázis mezőnevei elé a FIELD alias nevet kell megadni (lásd példa).

Fontos! Ha a kulcsmező számára adunk új értéket, az index értéke is megváltozik, az aktuális rekord az új „helyére kerül", a rekordmutató pedig azonnal felveszi ezt az értéket! Ezért ha több rekordnál egyidejűleg kívánjuk a kulcsértéket módosítani, a REPLACE parancs kiadása előtt a SET ORDER TO 0 utasítássort adjuk meg. Ennek hatására az értékadás szekvenciálisan történik, ugyanakkor minden nyitott indexállományban végrehajtódik a módosítás.

Hálózatban az aktuális rekordra vonatkozó REPLACE előtt az RLOCK(), hatókör vagy feltétel megadása esetén az FLOCK() függvénnyel, vagy már az állomány kizárólagos módon történő megnyitásával (USE ... EXCLUSIVE) zároljuk a módosítás alatti rekordot! Ha a mező értékét egy másik munkaterületen megnyitott adatbázisban kívánjuk módosítani, a rekordot RLOCK()-kal ott is zárolni kell!

Példák:

A REPLACE parancs egyszerű alkalmazása:

```
USE Sue NEW
APPEND BLANK
USE Dedi NEW
APPEND BLANK
//
REPLACE eset WITH Sue->ertek * db,;
   hely WITH Sue->hely,Sue->datum WITH DATE()
```

Ugyanez az értékadás a REPLACE használata nélkül:

```
FIELD->eset := Sue->ertek * FIELD->db
FIELD->hely := Sue->hely
Sue->datum := DATE()
```

Helye: CLIPPER.LIB

Lásd még: COMMIT, FLOCK(), RLOCK()

REPORT FORM

Adatok formázott megjelenítése

REPORT FORM <formanév>
[TO PRINTER] [TO FILE <állománynév>]
[<hatókör>] [WHILE <feltétel1>] [FOR <feltétel2>]
[PLAIN | HEADING <fejléc>] [NOEJECT] [SUMMARY]

<formanév> — a megjelenítés előre definiált formáját tartalmazó állomány (.frm) neve. Ha a kiterjesztést nem adjuk meg, az alapértelmezés .frm. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

TO PRINTER — a megjelenítés a nyomtatóra történik.

TO FILE <állománynév> — a formázott kiíratás a megadott nevű állományba történik. Az ASCII 12 (lapdobás) vezérlőkaraktereket nem veszi figyelembe. Ha a kiterjesztést nem adjuk meg, az alapértelmezés .txt. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

<hatókör> — az adatbázis megjeleníteni kívánt része. Alapértelmezése: ALL.

WHILE <feltétel1> — a megjelenítés mindaddig folyik, amíg a feltétel igaz.

FOR <feltétel2> — csak a feltételnek megfelelő rekordok kerülnek megjelenítésre.

PLAIN — hatására a megjelenítés lapdobások nélkül, valamint a dátum és a lapszám kihagyásával történik, a cím és a fejléc csak az első lapon jelenik meg.

HEADING <fejléc> — hatására a <fejléc> minden lap első sorában megjelenik. Ha a PLAIN és a HEADING egyidejűleg lett megadva, a PLAIN hatása érvényesül.

NOEJECT — nyomtatón történő megjelenítés esetén az első lap nyomtatása előtt nem dob lapot.

SUMMARY — megadása esetén csak a különböző összegfokozatokat tartalmazó sorok (alcsoportok, csoportok összege, mindösszesen) jelennek meg, az egyedi értékeket tartalmazó rekordok nem.

A REPORT FORM parancs segítségével az aktuális munkaterületen megnyitott adatbázis táblázatos vagy csoportosított formában való megjelenítése lehetséges. A megjelenítés formáját, valamint a lapok és az egyes oszlopok fejlécét a formát leíró .frm állomány tartalmazza, amelyet az RL.EXE segédprogrammal (ismertetése a szerző „*Clipper segédprogramok*“ című könyvében), vagy dBase III PLUS alatt hozhatunk létre. A megjelenítés a képernyőn, választhatóan a nyomtatón és/vagy egy megadott állományba történik. A képernyőn történő megjelenítés letiltása a REPORT FORM parancs előtt a SET CONSOLE OFF kapcsolóállítással lehetséges!

Ha a <formanév> nem tartalmaz meghajtót, illetve elérési útvonalat, és az állomány nem található az aktuális könyvtárban, keresése a SET PATH szerinti elérési útvonalon történik.

- A REPORT FORM megszakítási lehetőségét nekünk kell biztosítani a FOR feltételeként megadott INKEY() függvénnyel (lásd példa).
- A REPORT FORM figyelembe veszi a nyomtatás margójaként előzetesen beállított SET MARGIN értéket.
- Ha a lemezre írt állományban meg szeretnénk tartani a lapdobás (ASCII 12 kód) karaktereket, akkor a TO FILE paraméter helyett a következő példa szerint járjunk el:

```
SET PRINTER TO <allomanynev>
REPORT FORM <formanev> TO PRINTER
SET PRINTER TO
```

- Hálózatban a REPORT FORM parancs működése idején biztosítani kell, hogy más felhasználó ne változtathasson a kulcsértékeken, ellenkező esetben — a változtatástól függően — rekordok maradhatnak ki, vagy szerepelhetnek többszörösen.

TO FILE <állománynév> — a formázott kiíratás a megadott nevű állományba történik. Az ASCII 12 (lapdobás) vezérlőkaraktereket nem veszi figyelembe. Ha a kiterjesztést nem adjuk meg, az alapértelmezés .txt. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

<hatókör> — az adatbázis megjeleníteni kívánt része. Alapértelmezése: ALL.

WHILE <feltétel1> — a megjelenítés mindaddig folyik, amíg a feltétel igaz.

FOR <feltétel2> — csak a feltételnek megfelelő rekordok kerülnek megjelenítésre.

PLAIN — hatására a megjelenítés lapdobások nélkül, valamint a dátum és a lapszám kihagyásával történik, a cím és a fejléc csak az első lapon jelenik meg.

HEADING <fejléc> — hatására a <fejléc> minden lap első sorában megjelenik. Ha a PLAIN és a HEADING egyidejűleg lett megadva, a PLAIN hatása érvényesül.

NOEJECT — nyomtatón történő megjelenítés esetén az első lap nyomtatása előtt nem dob lapot.

SUMMARY — megadása esetén csak a különböző összegfokozatokat tartalmazó sorok (alcsoportok, csoportok összege, mindösszesen) jelennek meg, az egyedi értékeket tartalmazó rekordok nem.

A REPORT FORM parancs segítségével az aktuális munkaterületen megnyitott adatbázis táblázatos vagy csoportosított formában való megjelenítése lehetséges. A megjelenítés formáját, valamint a lapok és az egyes oszlopok fejlécét a formát leíró .frm állomány tartalmazza, amelyet az RL.EXE segédprogrammal (ismertetése a szerző „*Clipper segédprogramok*" című könyvében), vagy dBase III PLUS alatt hozhatunk létre. A megjelenítés a képernyőn, választhatóan a nyomtatón és/vagy egy megadott állományba történik. A képernyőn történő megjelenítés letiltása a REPORT FORM parancs előtt a SET CONSOLE OFF kapcsolóállítással lehetséges!

Ha a <formanév> nem tartalmaz meghajtót, illetve elérési útvonalat, és az állomány nem található az aktuális könyvtárban, keresése a SET PATH szerinti elérési útvonalon történik.

- A REPORT FORM megszakítási lehetőségét nekünk kell biztosítani a FOR feltételeként megadott INKEY() függvénnyel (lásd példa).
- A REPORT FORM figyelembe veszi a nyomtatás margójaként előzetesen beállított SET MARGIN értéket.
- Ha a lemezre írt állományban meg szeretnénk tartani a lapdobás (ASCII 12 kód) karaktereket, akkor a TO FILE paraméter helyett a következő példa szerint járjunk el:

```
SET PRINTER TO <allomanynev>
REPORT FORM <formanev> TO PRINTER
SET PRINTER TO
```

- Hálózatban a REPORT FORM parancs működése idején biztosítani kell, hogy más felhasználó ne változtathasson a kulcsértékeken, ellenkező esetben — a változtatástól függően — rekordok maradhatnak ki, vagy szerepelhetnek többszörösen.

Példák:

A formát tartalmazó állomány megadása közvetlenül a nevével, illetve változón keresztül:

```
LOCAL forma := "tabform"
USE Sue INDEX Sue NEW
REPORT FORM tabform TO PRINTER FOR ertek = "100";
   HEADING "ÉRTÉK 100"
REPORT FORM (forma) TO PRINTER FOR ertek != "100"
```

A megszakítási lehetőség beépítése:

```
#define K_ESC 27
USE Sue INDEX Sue NEW
REPORT FORM tabform FOR INKEY() != K_ESC
```

Helye: CLIPPER.LIB

Lásd még: LABEL FORM, LIST

RESTORE

Memóriaváltozók elmentett értékeinek visszahozatala

RESTORE FROM <állománynév> [ADDITIVE]

<állománynév> — az elmentett memóriaváltozó értékeket tartalmazó állomány neve. Ha kiterjesztést nem adunk meg, az alapértelmezés .mem. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

ADDITIVE — megadása esetén a program aktuális memóriaváltozói is megmaradnak, ellenkező esetben a visszatöltés előtt ezek törlődnek.

A RESTORE az előzőleg SAVE paranccsal kimentett *public* és *private* memóriaváltozókat, és azok akkori értékeit hozza vissza. *Tömbökkel* és *local* változókkal ezek a műveletek **nem** hajthatók végre! A változók érvényességi köre nem menthető ki, azok a RESTORE parancs kiadási helyének megfelelő érvényességgel private változók lesznek. Ha az elmentés és a visszatöltés között létrehozott egyéb változóértékeinket meg kívánjuk tartani, az ADDITIVE parancsot kell kiadnunk! Ebben az esetben az azonos nevű változók értékei felülíródnak (és ezek is private típusúak lesznek).

A *local* és *static* változókra a RESTORE parancs nincs hatással.

Példa:

Egy tipikus alkalmazás:

```
SAVE SCREEN TO kep1
SAVE ALL LIKE kep1valt TO k1valt
//
   <utasítások">
//
RESTORE FROM k1valt ADDITIVE
sorsz = "1"
```

Helye: CLIPPER.LIB

Lásd még: RESTORE, SAVE, RESTSCREEN(), SAVESCREEN()

RUN

DOS parancs vagy felhasználói program indítása

RUN | !* <parancs>

<parancs> — DOS parancs (beleértve a rezidens DOS parancsokat és a COMMAND.COM-ot is) vagy felhasználói program neve, amelyet a futó programból kívánunk indítani.

A RUN parancs hatására a megadott DOS parancs vagy felhasználói program kapja meg a vezérlést, annak végrehajtása, illetve lefutása után a vezérlés visszaadódik programunknak. A SET DEFAULT és a SET PATH értékek a RUN parancsnál nem érvényesek!

A ! indítási forma a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem ajánlott.

Fontos! Memóriarezidens programot ne indítsunk, mert a programunkba való visszatérés ebben az esetben nem biztosított!

Példák:

Bemutatjuk, hogyan használható a RUN parancs a MEMOREAD() és a MEMOWRIT() függvényekkel együtt memo-mezőnk saját szövegszerkesztővel történő használatára:

```
szerk = szovszer("KEDIT", "megj")
RETURN


FUNCTION szovszer( sajszerk, memomezo )
   IF MEMOWRIT("szoveg.tmp", &memomezo.)
      RUN (sajszerk + " szoveg.tmp")
      REPLACE &memomezo. WITH MEMOREAD("szoveg.tmp")
      ERASE szoveg.tmp
      RETURN .T.
   ELSE
      RETURN .F.
   ENDIF
```

Lehetőség van a DOS-ba történő kilépésre, és ott különböző parancsok kiadására, majd az EXIT parancs kiadása után a felhasználói programba visszatérésre:

```
RUN COMMAND
```

Helye: CLIPPER.LIB

SAVE

Memóriaváltozók értékeinek elmentése

SAVE TO <állománynév> [ALL [LIKE | EXCEPT <változók>]]

<állománynév> — a menteni kívánt memóriaváltozó értékeket tartalmazó állomány neve. Ha kiterjesztést nem adunk meg, az alapértelmezés .mem. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

ALL — az összes *public* és *private* memóriaváltozó elmentésre kerül.

ALL LIKE <változók> — csak a megadott *public* és *private* memóriaváltozók kerülnek elmentésre. A változónevek megadásánál használhatók a * és a ? csoportképző karakterek is.

ALL EXCEPT <változók> — a megadott változók **nem** kerülnek elmentésre, de az összes többi igen. A változónevek megadásánál használhatók a * és a ? csoportképző karakterek is.

A SAVE a parancskiadás helyén „látható" *public* és *private* memóriaváltozók értékét másolja egy .mem állományba. *Tömbök*, *local* és *static* változók értékei a SAVE paranccsal **nem** menthetők. A mentés a változók érvényességi körére nem terjed ki, így az a RESTORE paranccsal való visszatöltés után az ottani helyzetnek megfelelően változhat! A PRIVATE vagy a LOCAL deklarációk számára rejtett változók nem másolódnak ki.

Példák:

Az összes „látható" private és public változó kimentése a Sue.mem állományba:

```
PRIVATE mvegy := "1"
SAVE ALL TO Sue
```

Az mv-vel kezdődő összes private és public változó kimentése a Sue.mem állományba:

```
SAVE ALL LIKE mv* TO Sue
```

Az összes private és public változó kimentése a Sue.mem állományba, az mv-vel kezdődőek kivételével:

```
SAVE ALL EXCEPT mv* TO Sue
```

Helye: CLIPPER.LIB

Lásd még: LOCAL, PRIVATE, PUBLIC, RESTORE

SAVE SCREEN*

Az aktuális képernyőtartalom mentése/tárolása

SAVE SCREEN [TO <változó>]

TO <változó> — az aktuális képernyőtartalom karakteres értékként a <változó>-ba kerül elmentésre. A <változó> lehet mezőnév, local, static vagy tömbelem is. (Figyeljünk azonban arra, hogy tömböt, local és static változót a SAVE paranccsal nem lehet .mem állományba elmenteni!) Ha a <változó> még nem lett definiálva, vagy az adott helyen nem „látható", private memóriaváltozóként szerepel.

A SAVE SCREEN hatása a SVESCREEN() függvénnyel megegyező. A 0,0 és a MAXROW(),MAXCOL() közé eső képernyőtartalmat menti el a képernyőpufferbe, vagy a megadott változóba. A SAVE SCREEN paranccsal elmentett képernyőtartalmak a RESTORE SCREEN paranvvsal hozhatók vissza. Ha több képernyőt kívánunk menteni, akkor azokat különböző változónevekkel azonosítsuk!

A SAVE SCREEN a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem javasolt! Használjuk helyette a SAVESCREEN() függvényt, amellyel nemcsak teljes, hanem rész képernyőtartalmakat is el lehet menteni!

Fontos! A SAVE SCREEN, RESTORE SCREEN, SAVESCREEN() és RESTSCREEN() csak az alap (IBM PC memory mapped) képernyőmeghajtó esetén működik. Más esetekben hatástalan!

Példák:

Példa arra, hogyan lehet a képernyőtartalmat egy tömbelembe menteni:

```
STATIC kep[10]
SAVE SCREEN TO kep[1]
//
   <utasítások">
//
RESTORE SCREEN FROM kep[1]
```

A képernyőtartalmat egy adatbázismezőbe mentjük:

```
USE kepek INDEX kepnev NEW
APPEND BLANK
kepek->kepnev := "kep1"      // "a kép nevének elmentése"
SAVE SCREEN TO kepek->tart   // "a tartalom elmentése"
//
   <utasítások">
//
SEEK "kep1"                      // "a kívánt kép keresése"
RESTORE SCREEN FROM kepek->tart   // "kép visszatöltése"
```

Helye: CLIPPER.LIB

Lásd még: RESTORE, SAVE, RESTSCREEN(), SAVESCREEN()

SEEK

Indexkulcs szerinti keresés

SEEK <kifejezés>

<kifejezés> — a keresett indexkulcs, vagy annak része.

A SEEK az aktuálisan megnyitott indexállományban megkeresi azt az első rekordot, amely a <kifejezés>-ben megadott kulcsértékkel rendelkezik. Ha talált ilyet, akkor a rekordmutató felveszi ennek a rekordnak a sorszámát, a FOUND() értéke igaz (.T.); ha nem talált, akkor a rekordmutató értéke LASTREC()+1, a FOUND() hamis (.F.), az EOF() igaz (.T.). Ha a SOFTSEEK ON állásban van, és a <kifejezés> szerinti kulcsérték nem létezik, akkor a rekordmutató

a <kifejezés> értékénél nagyobb legközelebbi rekordra mutat, a FOUND() értéke hamis (.F.). Az EOF() ebben az esetben csak akkor hamis (.F.), ha a keresett <kifejezés>-nél nem létezik nagyobb az állományban.

A SET EXACT nincs hatással a SEEK parancs működésére. A megadott <kifejezés>-sel pontosan megegyező kulcs keresésére lásd a példát.

Példák:

Keresés egyszerű SEEK paranccsal:

```
USE Sue INDEX Dedi NEW
SEEK "0530"
? FOUND(), EOF(), RECNO()      // Eredmény: .T. .F. 117
```

Példa a megadott <kifejezés>-sel pontosan megegyező kulcs keresésére:

```
USE Sue INDEX Dedi NEW
IF Pontosan("860530")
   ? "Megtaláltam!"
ELSE
   ? "Nem találom!"
ENDIF
RETURN
FUNCTION Pontosan( kod )
   SEEK PADR(kod, LEN(&(INDEXKEY(0))))
   RETURN (FOUND())
```

Helye: CLIPPER.LIB

Lásd még: SET SOFTSEEK, EOF(), FOUND()

SELECT

Az aktuális munkaterület cseréje

SELECT <munkaterület sorszám> | <alias név>

<munkaterület sorszám> — 0 és 250 közé eső szám lehet. Megadható közvetlen numerikus értékként, vagy zárójelbe helyezett numerikus változóként is.

<alias név> — a nyitott adatbázist tartalmazó munkaterület azonosító neve.

A SELECT az aktuális munkaterületek váltására szolgál. A Clipper maximálisan 250 munkaterületet képes azonosítani. A szükséges munkaterület kiválasztása annak sorszámával, vagy alias nevével lehetséges. Az ALIAS nevet az adatbázis megnyitásakor lehet megadni, a USE parancs ALIAS kulcsszava után. Ha ezt nem adjuk meg, az <alias név> automatikusan az adatbázis neve lesz.

A SELECT 0 az első (következő) szabad munkaterületet adja, ugyanazt eredményezi, mint a USE ... NEW parancs. Használatuk esetén azonban legyünk óvatosak, mert egy későbbi direkt sorszámmal történő munkaterület kijelölés felülírja ezeket! (Ilyen vegyes használat esetén a SELECT() függvénnyel előbb vizsgáljuk meg, hogy a munkaterület szabad-e.)

A munkaterületek azonosítása jól megválasztott alias nevekkel a programozást igen hatékonyá teszi. Többek között lehetőséget biztosít arra is, hogy bizonyos értékek vizsgálatához nem kell a munkaterületet váltani, hanem az alias név felhasználásával ez közvetlenül elérhető, például:

```
SELECT Sue
? EOF()
SELECT Lise
```

helyett elegendő a következőt beírni:

```
? Sue->(EOF())
```

Példák:

Adatbázisok különbözőképpen történő megnyitása, és ezek munkaterületének elérése:

```
SELECT 1
USE Sue
SELECT 2
USE Dedi ALIAS Sajat
USE Lise NEW          // "a 3. munkaterület"
SELECT 0
USE Kate              // "a 4. munkaterület"
```

az elérések:

```
SELECT Sue      // "az 1.munkaterület"
   <utasítások">
SELECT Sajat    // "a 2. munkaterület"
   <utasítások">
SELECT 4
   <utasítások">
```

Vegyes kijelölés esetén a szabad munkaterület sorszámának megállapítása:

```
USE Mary NEW
ujmter := SELECT()
   <utasítások">
SELECT (ujmter+1)
USE Jude
```

Helye: CLIPPER.LIB

Lásd még: SET INDEX, USE, ALIAS(), SELECT()

SET ALTERNATE

Megjelenítés állományban

SET ALTERNATE TO [<állománynév>]
SET ALTERNATE on|OFF|<logikai kifejezés>

TO <állománynév> — egy szabvány ASCII állományt nyit meg. A kiterjesztés alapértelmezése .txt. Az <állománynév> tetszőlegesen tartalmazhat meghajtó és elérési útvonal meghatározást, valamint kiterjesztést. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karak-

teres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. Ha a megadott nevű állomány már létezik, az figyelmeztetés nélkül felülíródik!

ON — engedélyezi a megjelenítés átirányátását az előzőleg megnyitott állományba.

OFF — felfüggeszti a megjelenítés átirányítását, de az állományt nem zárja le. Ez az alapértelmezés.

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően az átirányítás megvalósul, illetve felfüggesztődik. A kifejezés igaz (.T.) értéke az ON, hamis (.F.) értéke az OFF kapcsolóállásnak felel meg.

A SET ALTERNATE parancs segítségével a LIST, REPORT FORM, LABEL FORM, ? és ?? parancsok kiadása után képernyőn megjelenő információk irányíthatók át a megnyitott szövegállományba. E parancsokban a TO FILE <állománynév> megadása a SET ALTERNATE paranccsal azonos hatású. A teljes képernyős parancsok, mint a @...SAY, a SET ALTERNATE-tel nem irányítható át. Ehhez a SET PRINTER TO <állománynév> és a SET DEVICE TO PRINTER parancsokat kell alkalmazni!

A SET ALTERNATE TO egy DOS szövegállományt hoz létre, amely az aktuális munkaterülettől független. Egyszerre csak egy alternatív állomány lehet nyitva. Lezárását a CLOSE ALTERNATE, CLOSE ALL, vagy az állománynév nélkül megadott SET ALTERNATE TO parancsokkal végezhetjük.

Az on|OFF kapcsolók az állományba írás végrehajtását, illetve felfüggesztését vezérlik.

A megjelenítendő információk a képernyőn az átirányítás esetén is láthatók. Ennek letíltása a SET CONSOLE OFF paranccsal lehetséges.

Példa:

Létrehozunk egy alternatív állományt, majd a ? paranccsal kiíratjuk az adatbázis rekordok két mezőjének tartalmát.

```
SET ALTERNATE TO Lista
SET ALTERNATE ON
USE Sue NEW
DO WHILE !EOF()
   ? eset, datum
   SKIP
ENDDO
SET ALTERNATE OFF
CLOSE ALTERNATE
CLOSE Sue
```

Helye: CLIPPER.LIB

Lásd még: CLOSE, SET CONSOLE, SET PRINTER, FCREATE(), FOPEN(), FWRITE()

SET BELL

Jelzőhang be/kikapcsolása

SET BELL on|OFF|<logikai kifejezés>

ON — a jelzőhang bekapcsolása.

OFF — a jelzőhang kikapcsolása. Ez az alapértelmezés.

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően a jelzőhang be- vagy kikapcsolódik. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

A SET BELL ON hatására a jelzőhang a következő esetekben szólal meg:

- a GET mező utolsó karakterének megadása után;
- ha a GET mezőbe hibás adatot írtunk be. Az ellenőrzést a GET PICTURE és RANGE paraméterei szerint végzi. A VALID paraméter nincs hatással a jelzőhangra.

A jelzőhang megszólaltatható a ?? CHR(7) és a TONE() függvénnyel is. Ez utóbbi segítségével szabályozható a hang magassága és hossza is.

Helye: CLIPPER.LIB

Lásd még: @...GET,SET CONFIRM, CHR(), TONE()

SET CENTURY

Az évszázad kijelzése a dátumokban

SET CENTURY on|OFF|<logikai kifejezés>

ON állásban a dátumoknál kijelzésre kerül az évszázad is.

OFF állásban csak az évszám utolsó két jegye jelenik meg. Ez az alapértelmezés.

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően az évszázad megjelenik vagy nem. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

A SET CENTURY módosítja a megjelenő dátum formátumát olymódon, hogy a SET DATE szerint beállított dátumformában az évszázadok is megjelennek, azaz az évet négyjegyű szám jelzi.

A program által számított dátumok kijelzésére a SET CENTURY nincs hatással. A Clipper által érvényesnek elfogadott dátum 0100.01.01-től 2999.12.31-ig terjed.

Példa:

A SET CENTURY parancs működésének hatása:

```
SET CENTURY OFF
? DATE()              // Eredmény: 17/01/91
SET CENTURY ON
? DATE()              // Eredmény: 17/01/1991
```

SET BELL

Jelzőhang be/kikapcsolása

SET BELL on|OFF|<logikai kifejezés>

ON — a jelzőhang bekapcsolása.

OFF — a jelzőhang kikapcsolása. Ez az alapértelmezés.

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően a jelzőhang be- vagy kikapcsolódik. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

A SET BELL ON hatására a jelzőhang a következő esetekben szólal meg:

- a GET mező utolsó karakterének megadása után;
- ha a GET mezőbe hibás adatot írtunk be. Az ellenőrzést a GET PICTURE és RANGE paraméterei szerint végzi. A VALID paraméter nincs hatással a jelzőhangra.

A jelzőhang megszólaltatható a ?? CHR(7) és a TONE() függvénnyel is. Ez utóbbi segítségével szabályozható a hang magassága és hossza is.

Helye: CLIPPER.LIB

Lásd még: @...GET,SET CONFIRM, CHR(), TONE()

SET CENTURY

Az évszázad kijelzése a dátumokban

SET CENTURY on|OFF|<logikai kifejezés>

ON állásban a dátumoknál kijelzésre kerül az évszázad is.

OFF állásban csak az évszám utolsó két jegye jelenik meg. Ez az alapértelmezés.

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően az évszázad megjelenik vagy nem. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

A SET CENTURY módosítja a megjelenő dátum formátumát olymódon, hogy a SET DATE szerint beállított dátumformában az évszázadok is megjelennek, azaz az évet négyjegyű szám jelzi.

A program által számított dátumok kijelzésére a SET CENTURY nincs hatással. A Clipper által érvényesnek elfogadott dátum 0100.01.01-től 2999.12.31-ig terjed.

Példa:

A SET CENTURY parancs működésének hatása:

```
SET CENTURY OFF
? DATE()              // Eredmény: 17/01/91
SET CENTURY ON
? DATE()              // Eredmény: 17/01/1991
```

Helye: CLIPPER.LIB

Lásd még: SET DATE, SET EPOCH, CTOD(), DATE(), DTOC()

SET COLOR*

Képernyőszínek meghatározása

SET COLOR|COLOUR TO [[<normál>] [,<kiemelt>] [,<keret>] [,<háttér>] [,<nem kiválasztott>] | <karakteres kifejezés>]

<normál> — a normál információmegjelenítés színe. Ezt használja minden adatkiíró utasítás, így a @...PROMPT, a @...SAY, a ?, valamint az ACHOICE(), A DBEDIT() ÉS A MEMOEDIT() függvények.

<kiemelt> — az aktuális mező jelzésére használt színek: GET-nél az éppen bekérés alatt álló mező, a MENU TO, DBEDIT() és ACHOICE() függvényeknél az aktuális menüpont stb.

<keret> — a képernyő keretének színe. Erre a területre az írás nincs megengedve, így a szín meghatározásánál is csak az első kódot veszi figyelembe.

<háttér> — nem használt kód, csak a korábbi verziókkal való kompatibilitás érdekében maradt meg.

<nem kiválasztott> — a GET utasítás nem aktuális mezőinek színe. Az éppen aktuális GET-mező a <kiemelt> színnel jelenik meg. Ha ezt a paramétert nem használjuk, a rendszer csak a kiemelt színnel dolgozik.

<karakteres kifejezés> — az előzőekben felsorolt paraméterek megadhatók egyetlen karakteres kifejezésben is, aminek tartalma lehet idézőjelek közé tett karaktersorozat, vagy makró-kifejezés. A kifejezést zárójelbe kell tenni.

A SET COLOR TO alapértelmezése W/N, N/W, N, N, N/W.

A SET COLOR a SETCOLOR() függvény szinonímája, az utóbbi azonban a részképernyő színbeállítását, valamint az aktuális színértékek megállapítását is lehetővé teszi. A megadható színeket két kód határozza meg: az első mutatja, hogy milyen színnel ír, a második, hogy milyen alapra. A két kód közé / jelet kell tenni. Szóköz, illetve nem-display karakterek esetében csak az alap színe látszik.

A kiírás színének kódjához illesztett "+" hatására az illető szín fényesebb lesz, "*" hatására pedig a megadott színnel megjelenített szöveg villogni fog.

Minden szín megadható betűvel, vagy számmal, (lásd táblázat) azonban ezek vegyesen nem használhatók. A számmal való megadási lehetőség csak a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazását kevésbé ajánljuk. Mindegyik színbeállítási lehetőség opcionális: tetszőlegesen bármelyik (vagy mind) elhagyható. Ekkor az alapértelmezés lép érvénybe. Monochrom monitoron a táblázat „monochrom képernyőn" oszlopában megadott hatás jelentkezik.

```
      @ sor, oszl SAY "Kérem a jelszót: " GET mitadbe
      READ
      //
      IF LASTKEY == 27
         SET COLOR TO (aktszin)
         RETURN .F.
      ELSEIF mitadbe == jelszo
         SET COLOR TO (aktszin)
         RETURN .T.
      ELSE
         szaml++
      ENDIF
   ENDDO
   //
   SET COLOR TO (aktszin)
   RETURN .F.
```

Helye: CLIPPER.LIB

Lásd még: @...GET, @...SAY, ISCOLOR(), SETCOLOR()

SET CONFIRM

Kilépőbillentyű használata a GET befejezéséhez

SET CONFIRM on|OFF|<logikai kifejezés>

ON — hatására a @...GET parancs csak a kilépőbillentyűk valamelyikének lenyomására fejezi be a mezőérték beolvasását.

OFF — a mezőérték beírása után automatikusan a következő utasítás végrehajtása következik be. Ez az alapértelmezés.

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően a kilépőbillentyű lenyomására várakozás bekövetkezik vagy nem. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

A kilépőbillentyűk: Ctrl-C, PgUp, Ctrl-W, PgDn, Esc, Return.

Helye: CLIPPER.LIB

Lásd még: @...GET, READ, SET BELL

SET CONSOLE

A képernyőre írás engedélyezése/letíltása

SET CONSOLE ON|off|<logikai kifejezés>

ON — a képernyőre írás engedélyezve van. Ez az alapértelmezés.

OFF — a képernyőre írás letíltása.

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően a képernyőre írás engedélyezve van, vagy nincs. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

A SET CONSOLE kapcsolóállítással szabályozhatjuk a konzolparancsok képernyőre írását, illetve letíltását. Konzolparancsoknak nevezzük azokat a parancsokat, amelyek a megjelenítéshez nem használják fel a képernyő sor és oszlopértékét (ACCEPT, INPUT, WAIT, ?, ??, ... TO PRINT, REPORT FORM, LABEL FORM). Ugyanakkor ezzel a paranccsal nem tiltható le a @..., CLEAR SCREEN parancsok hatása.

Input parancsoknál (ACCEPT, INPUT, WAIT) a SET CONSOLE OFF hatására a billentyűzetről beírt értékek nem látszanak a képernyőn, de a program elfogadja azokat.

A teljes képernyős (@...) parancsok szerinti megjelenítések csak a SET DEVICE TO PRINT és a SET PRINTER TO átirányítás után tilthatók le a képernyőről.

Példa:

Egyszerű példa a képernyőn való megjelenítés letíltására:

```
USE Sue NEW
SET CONSOLE OFF
REPORT FORM Sue TO PRINTER
SET CONSOLE ON
```

Helye: CLIPPER.LIB

Lásd még: SET DEVICE

SET CURSOR

A kurzor jelenlétének engedélyezése/tíltása

SET CURSOR ON|off|<logikai kifejezés>

ON — a kurzor jelenlétének engedélyezése (alapértelmezés).

OFF — a kurzor letíltása (nem látszik).

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően a kurzor jelenléte engedélyezve van, vagy nincs. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

A SET CURSOR kapcsoló állításával a kurzort láthatóvá tehetjük, illetve eltüntethetjük. OFF állapot esetén a billentyűzetről történő adatbevitel és a képernyős megjelenítés hatástalan. Ugyanakkor a ROW() és a COL() értékek ugyanúgy változnak, mintha a kurzor látható lenne.

Ennek a parancsnak a segítségével kiküszöbölhetjük azt, hogy programunk futása közben a kurzor „teljesen céltalanul", funkció nélkül a képernyő különböző helyén villogjon. A kurzorra ideális esetben csak a képernyős megjelenítést igénylő utasításoknál van szükség.

Példa:

Egy tipikus alkalmazás:

```
USE Sue NEW
@ SAY 12, 12 "Adatok másolása folyik!"
SET CURSOR OFF
COPY TO Lise
SET CURSOR ON
```

Helye: CLIPPER.LIB

Lásd még: SET CONSOLE, SETCURSOR(), SETPOS()

SET DATE

A dátumforma meghatározása

SET DATE [TO] AMERICAN|ansi|british|french|gerınan|italian|japan|usa
SET DATE FORMAT [TO] <dátumforma>

<dátumforma> — karakteres kifejezés, amelynek értékével direkt módon határozhatjuk meg a használni kívánt dátum formáját. Legfeljebb 12 karakter hosszú lehet. A <dátumforma>-ban az év, hónap és nap helyének meghatározására az *y*, *m* és *d* karakterek szolgálnak. Az ezektől eltérő karaktereket a dátumtípusú értékek kijelzésekor változatlan formában jeleníti meg.

A SET DATE hatása a teljes programra érvényes, függetlenül a parancs kiadásának helyétől.

A megadható dátumformák felépítése:

AMERICANmm/dd/yy

ANSIyy.mm.dd

BRITISHdd/mm/yy

FRENCHdd/mm/yy

GERMANdd.mm.yy

ITALIANdd-mm-yy

JAPANyy/mm/dd

USAmm-dd-yy

Példák:

Saját dátumformátum megadása:

```
SET DATE FORMAT "yyyy:mm:dd"
```

Ha a dátumformát a felhasználói program módosítása nélkül DOS-ból kívánjuk a későbbiekben változtatni, azt a GETENV() függvény felhasználásával az alábbi módon tehetjük meg:

```
C>SET CLIP_DATE=yyyy.mm.dd      //DOS-ban adva meg
```

Példa:

Egy tipikus alkalmazás:

```
USE Sue NEW
@ SAY 12, 12 "Adatok másolása folyik!"
SET CURSOR OFF
COPY TO Lise
SET CURSOR ON
```

Helye: CLIPPER.LIB

Lásd még: SET CONSOLE, SETCURSOR(), SETPOS()

SET DATE

A dátumforma meghatározása

SET DATE [TO] AMERICAN|ansi|british|french|german|italian|japan|usa
SET DATE FORMAT [TO] <dátumforma>

<dátumforma> — karakteres kifejezés, amelynek értékével direkt módon határozhatjuk meg a használni kívánt dátum formáját. Legfeljebb 12 karakter hosszú lehet. A <dátumforma>-ban az év, hónap és nap helyének meghatározására az *y*, *m* és *d* karakterek szolgálnak. Az ezektől eltérő karaktereket a dátumtípusú értékek kijelzésekor változatlan formában jeleníti meg.

A SET DATE hatása a teljes programra érvényes, függetlenül a parancs kiadásának helyétől.

A megadható dátumformák felépítése:

AMERICANmm/dd/yy

ANSIyy.mm.dd

BRITISHdd/mm/yy

FRENCHdd/mm/yy

GERMANdd.mm.yy

ITALIANdd-mm-yy

JAPANyy/mm/dd

USAmm-dd-yy

Példák:

Saját dátumformátum megadása:

```
SET DATE FORMAT "yyyy:mm:dd"
```

Ha a dátumformát a felhasználói program módosítása nélkül DOS-ból kívánjuk a későbbiekben változtatni, azt a GETENV() függvény felhasználásával az alábbi módon tehetjük meg:

```
C>SET CLIP_DATE=yyyy.mm.dd      //DOS-ban adva meg
```

Az így definiált dátumformát például az alábbi függvénnyel használhatjuk programunkban:

```
FUNCTION datforma
   SET DATE FORMAT TO GETENV("CLIP_DATE")
   RETURN NIL
```

Helye: CLIPPER.LIB

Lásd még: SET CENTURY, SET EPOCH, CTOD(), DATE(), DTOC()

SET DECIMALS

A tizedesjegyek számának meghatározása

SET DECIMALS TO [<numerikus kifejezés>]

<numerikus kifejezés> — a maximálisan megjelenítésre kerülő tizedesjegyek száma. Ha nem adunk meg semmit, értéke 2.

SET DECIMALS TO érték nélküli megadása a SET DECIMALS TO 0-val egyenértékű.

A SET DECIMALS parancs határozza meg a tizedesjegyek számát a számítások és a numerikus függvények eredményeinek megjelenítése során. A SET FIXED paranccsal együtt használva biztosítja az eredmények egységes hosszúságú kiíratását. Ha a FIXED OFF, akkor az EXP(), LOG(), SQRT() függvények és az osztás művelete után csak az értékes tizedesjegyek íródnak ki; ha a FIXED ON, mindig pontosan a SET DECIMALS szerint meghatározott számú tizedesjegy jelenik meg.

A SET DECIMALS és a SET FIXED csak az eredmények megjelenítésére van hatással, a számítások pontosságára nincs!

A megjelenítés hosszának szabályozására szolgál a @...SAY és a @...GET parancsok PICTURE paramétere, valamint a TRANSFORM() függvény is.

Példa:

A SET DECIMALS parancs hatása az értékek megjelenítésére:

```
SET FIXED ON
SET DECIMALS TO 3
? 1/2        // Eredmény: 0.500
? 1/3        // Eredmény: 0.333
SET FIXED OFF
SET DECIMALS TO 4
? 1/2        // Eredmény: 0.5
? 1/3        // Eredmény: 0.3333
```

Helye: CLIPPER.LIB

Lásd még: SET FIXED, TRANSFORM()

A SET DELETED hatását munkaterülettől függetlenül kifejti. A törlésre kijelölt rekordra GOTO-val történő direkt ráállás esetén azonban a rekord a kapcsoló ON állásában is feldolgozható. A SET DELETED helyzete nincs hatással az INDEX és REINDEX parancsokra (ezek minden rekordot feldolgoznak).

Ha csak egy munkaterületen kívánjuk kihagyni a feldolgozásból a törölt rekordokat, azt a kiválasztott munkaterületen a SET FILTER TO !DELETED() utasítással érhetjük el.

Példák:

A SET DELETED hatása:

```
USE Sue NEW
? LASTREC()          // Eredmény: 86
//
DELETE RECORD 17
COUNT TO darab
? darab              // Eredmény: 86
//
SET DELETED ON
COUNT TO darab
? darab              // Eredmény: 85
```

Helye: CLIPPER.LIB

Lásd még: DELETE, RECALL, SET FILTER, DELETED()

SET DELIMITERS

Mezőhatároló jelek definiálása

SET DELIMITERS on|OFF|<logikai kifejezés>
SET DELIMITERS TO [<határolójelek>|DEFAULT]

ON — a GET-ekben megjeleníti a mezőhatároló jeleket.

OFF — a határoló jelek nem jelennek meg (alapértelmezés).

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően a mezőhatároló jelek megjelenítése engedélyezve van, vagy nincs. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

TO <határolójelek> — egy vagy két karakter, amely a GET mező elején és végén jelenik meg. Ha nincs megadva <határolójel>, vagy a DEFAULT paramétert adtuk meg, a mezők elején és végén kettőspont jelenik meg.

SET DELIMITERS ON esetén a @...GET parancsban a változó számára meghatározott hossz kettővel nő. A határolójeleket a TO <határolójel> paraméterrel adhatjuk meg. Ha a mező elején vagy végén levő határolójelet el akarjuk nyomni, annak a helyére szóközt kell írni. A határolójelek nem jelennek meg, ha az aktuális GET inverz, vagy kiemelt színű és az INTENSITY ON kapcsolót állítottuk be.

A SET DELETED hatását munkaterülettől függetlenül kifejti. A törlésre kijelölt rekordra GOTO-val történő direkt ráállás esetén azonban a rekord a kapcsoló ON állásában is feldolgozható. A SET DELETED helyzete nincs hatással az INDEX és REINDEX parancsokra (ezek minden rekordot feldolgoznak).

Ha csak egy munkaterületen kívánjuk kihagyni a feldolgozásból a törölt rekordokat, azt a kiválasztott munkaterületen a SET FILTER TO !DELETED() utasítással érhetjük el.

Példák:

A SET DELETED hatása:

```
USE Sue NEW
? LASTREC()          // Eredmény: 86
//
DELETE RECORD 17
COUNT TO darab
? darab              // Eredmény: 86
//
SET DELETED ON
COUNT TO darab
? darab              // Eredmény: 85
```

Helye: CLIPPER.LIB

Lásd még: DELETE, RECALL, SET FILTER, DELETED()

SET DELIMITERS

Mezőhatároló jelek definiálása

SET DELIMITERS on|OFF|<logikai kifejezés>
SET DELIMITERS TO [<határolójelek>|DEFAULT]

ON — a GET-ekben megjeleníti a mezőhatároló jeleket.

OFF — a határoló jelek nem jelennek meg (alapértelmezés).

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően a mezőhatároló jelek megjelenítése engedélyezve van, vagy nincs. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

TO <határolójelek> — egy vagy két karakter, amely a GET mező elején és végén jelenik meg. Ha nincs megadva <határolójel>, vagy a DEFAULT paramétert adtuk meg, a mezők elején és végén kettőspont jelenik meg.

SET DELIMITERS ON esetén a @...GET parancsban a változó számára meghatározott hossz kettővel nő. A határolójeleket a TO <határolójel> paraméterrel adhatjuk meg. Ha a mező elején vagy végén levő határolójelet el akarjuk nyomni, annak a helyére szóközt kell írni. A határolójelek nem jelennek meg, ha az aktuális GET inverz, vagy kiemelt színű és az INTENSITY ON kapcsolót állítottuk be.

Példa:

A határolójelek különböző formában történő megadási lehetőségei:

```
LOCAL valt1 := SPACE(5), valt2 := SPACE(5)
SET DELIMITERS ON
SET DELIMITERS TO "[]"
@ 5, 12 SAY "Elsö adat: " GET valt1    // [xxxxx]
SET DELIMITERS TO ": "
@ 6, 12 SAY "Adat2: " GET valt2    // :xxxxx
READ
```

Helye: CLIPPER.LIB

Lásd még: @...GET, SET INTENSITY

SET DEVICE

A megjelenítés irányítása képernyőre vagy nyomtatóra

SET DEVICE TO SCREEN|printer

SCREEN — a @...SAY paranccsal meghatározott információt a SET PRINTER és a SET CONSOLE állapotától függetlenül a képernyőre küldi (alapértelmezés).

PRINTER — a @...SAY paranccsal meghatározott információt a SET PRINTER TO által meghatározott meghajtóra küldi, ami lehet lokális, vagy hálózati nyomtató, illetve lemezállomány.

A SET DEVICE parancs a @...SAY szerinti megjelenítést a képernyőre vagy a nyomtatóra küldi. Ha a PRINTER paramétert alkalmazzuk, a megjelenítés a nyomtatóra történik, a képernyőre nem. A nyomtatásnál figyelembe veszi az aktuális SET MARGIN értéket. Nyomtatóra irányítás esetén a Clipper automatikusan lapdobást (EJECT) végez, ha a nyomtatandó sorérték kisebb az aktuális PROW() értéknél: ekkor a PROW() és PCOL() 0 értéket vesz fel. A PROW() és PCOL() értékek változtatására a SETPRC() függvény alkalmazható.

Ha a megjelenítést lemezállományba kívánjuk irányítani a SET DEVICE TO <állománynév> formát alkalmazzuk a SET DEVICE TO PRINTER utasítás kiadásával együtt.

Példák:

Megjelenítés a nyomtatón:

```
SET DEVICE TO PRINTER
@ 2, 10 SAY "Ez a nyomtatóra megy"
EJECT
```

Átirányítás lemezállományba:

```
SET  PRINTER TO Nyomt.txt
SET  DEVICE TO PRINTER
@ 10, 10 SAY "Lemezállomány neve: Nyomt.txt"
@ 11,10 SAY DATE()
SET PRINTER TO       // "a lemezállomány lezárása"
SET  DEVICE TO SCREEN
```

Helye: CLIPPER.LIB

Lásd még: @...SAY, EJECT, SET PRINTER, PCOL(), PROW(), SETPRC()

SET EPOCH

A dátum századokkal való kiegészítése

SET EPOCH TO <bázisév>

<bázisév> — az az évszám, amelyhez viszonyítjuk a két számjeggyel megadott évek megjelenítését. Az alapértelmezés 1900.

A SET EPOCH a csak két számjeggyel megadott évek évszázadokkal való automatikus kiegészítésére szolgál. Ha az évszám utolsó két számjegye nagyobb, vagy egyenlő a bázisévben megadottnál, akkor a bázisév szerinti évszázad jelenik meg, ha kisebb, akkor a következő évszázad.

A Clipper a dátumokat 0100.01.01-től 2999.12.31-ig tudja értelmezni.

Példa:

A SET EPOCH hatása:

```
SET DATE FORMAT TO "yyyy.mm.dd"
? CTOD("1955.10.14")      // Eredmény: 1955.10.14
? CTOD("86.05.30")        // Eredmény: 1986.05.30
? CTOD("03.12.08")        // Eredmény: 1903.12.08
//
SET EPOCH TO 1950
? CTOD("1955.10.14")      // Eredmény: 1955.10.14
? CTOD("86.05.30")        // Eredmény: 1986.05.30
? CTOD("03.12.08")        // Eredmény: 2003.12.08
```

Helye: CLIPPER.LIB

Lásd még: SET CENTURY, SET DATE, CTOD(), DATE(), DTOC()

SET ESCAPE

READ-ből kilépés az ESCAPE billentyűvel

SET ESCAPE ON|off|<logikai kifejezés>

ON — hatására az ESCAPE billentyű lenyomásával ki tudunk lépni az aktuális READ parancsból (alapértelmezés).

OFF — az ESCAPE billentyű lenyomása nincs hatással a READ parancsból való kilpésre.

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően az ESCAPE billentyű hatással van a READ-ből való kilépésre, vagy nincs. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

ON állapotban az ESCAPE lenyomásával kilépünk az aktuális READ-ből. Az előző READ óta beírt GET értékek elvesznek, a RANGE, illetve VALID paraméterekkel megadott ellenőrző feltételek törlődnek. A SET KEY paranccsal az ESCAPE billentyű funkciója megváltoztatható, ekkor a SET ESCPAE parancs hatástalan.

Helye: CLIPPER.LIB

Lásd még: READ, SET KEY, SETCANCEL()

SET EXACT*

Karakteres kifejezések pontos összehasonlítása

SET EXACT on|OFF|<logikai kifejezés>

ON — a karakteres kifejezések pontos összehasonlítását végzi, beleértve a hosszt is.

OFF — normál karakteres összehasonlítást végez.

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően az összehasonlítás pontosan, vagy normál módon történik. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

A SET EXACT paranccsal határozzuk meg, hogy két karakteres kifejezés összehasonlítása milyen módon menjen végbe. Az összehasonlításhoz az =, >, <, =>, =< operátorokat használhatjuk. SET EXACT OFF esetén az összehasonlítás a következő szabály szerint történik:

- ha a jobb oldalon üres kifejezés ("") van, az eredmény igaz (.T.);
- ha LEN(jobboldal) nagyobb, mint LEN(baloldal), az eredmény hamis (.F.);
- egyébként a bal oldaltól kezdve karakterenként összehasonlítja a két kifejezést, és ha a jobboldal minden karaktere megegyezik a baloldal minden karakterével, az eredmény igaz (.T.), egyébként hamis (.F.).

SET EXACT ON esetén az összehasonlítás az == operátornak megfelelő, amely szerint két karakteres kifejezés akkor egyezik meg, ha a kiegészítő szóközöktől megszabadított két karaktersztring minden karaktere azonos.

A SET EXACT a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem javasolt! A pontos összehasonlításhoz használjuk a == paramétert!

A SET EXACT parancs csak az operátorokkal való összehasonlításnál érvényes. A SEEK és a FIND parancsoknál hatástalan. Ha pontos érték szerinti keresést kívánunk végezni, használjuk a SEEK parancsnál példaként bemutatott Pontosan() függvényt.

Példák:

Az összehasonlítások eredményei a SET EXACT állásától függően:

```
SET EXACT OFF
? "987" = "98765"          // Eredmény: .F.
? "98765" = "987"          // Eredmény: .T.
? "987" = ""               // Eredmény: .T.
? "" = "98765"             // Eredmény: .F.
? "987" = "987  "          // Eredmény: .F.
```

```
//
SET EXACT ON
? "987" = "98765"          // Eredmény: .F.
? "98765" = "987"          // Eredmény: .F.
? "987" = ""               // Eredmény: .F.
? "" = "98765"             // Eredmény: .F.
? "987" = "987  "          // Eredmény: .T.
```

Helye: CLIPPER.LIB

Lásd még: SEEK

SET EXCLUSIVE*

Állomány kizárólagos vagy közös használatra megnyitása

SET EXCLUSIVE ON|off|<logikai kifejezés>

ON — az állomány (.dbf, .dbt, .ntx, .ndx) kizárólagos használatra lesz megnyitva (alapértelmezés).

OFF — az állomány közös használatra lesz megnyitva.

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően az állomány kizárólagos vagy közös használatra lesz megnyitva. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

Hálózatban a SET EXCLUSIVE parancs akkor fejti ki hatását, ha a USE parancshoz nem adjuk meg sem az EXCLUSIVE, sem a SHARED paramétert. A hatás kiterjed mind az adatbázis, mind az index, mind a memo állományra. Ha az állomány kizárólagosként lett megnyitva, azt más felhasználó nem tudja elérni mindaddig, amíg egy CLOSE paranccsal azt le nem zárjuk. Ebben az esetben az állomány és a rekord zárolásának nincs jelentősége.

Ha a SET EXCLUSIVE ON van érvényben, akkor minden olyan állomány kizárólagosként lesz megnyitva, amelynél a USE parancsban nincs feltüntetve a SHARED paraméter. A kizárólagos megnyitásra egyébként csak a PACK, REINDEX és ZAP parancsok esetén van szükség!

Ha a SET EXCLUSIVE OFF van érvényben, minden olyan állomány közös használatra nyílik meg, amelynél a USE parancsban nincs feltüntetve az EXCLUSIVE paraméter. A többi felhasználó által történő elérést az RLOCK() és FLOCK() függvényekkel ellenőrizni kell!

A SET EXCLUSIVE a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem javasolt! Használjuk helyette a USE parancsban az EXCLUSIVE vagy a SHARED paramétert!

Ha egy kizárólagosan megnyitott állományt a hálózatban egy másik felhasználó is meg akarja nyitni, hibajelzés következik be, és a NETERR() függvény értéke igaz (.T.) lesz. Ezért az állományok megnyitása előtt célszerű a NETERR() értékét megvizsgálni!

Példa:

Az állomány megnyitása előtti vizsgálat:

```
DO WHILE NETERR()
   USE Sue NEW
   ? "Az állományt másik állomás használja!"
   ? "Kis türelmet – vagy állíts le!"
ENDDO
```

Helye: CLIPPER.LIB

Lásd még: USE, FLOCK(), NETERR(), RLOCK()

SET FILTER

Szűrőfeltétel megadása

SET FILTER TO [<feltétel>]

<feltétel> — logikai kifejezés, amely meghatározza, hogy az aktuális munkaterületen megnyitott adatbázis mely rekordjai legyenek figyelembe véve.

A SET FILTER TO feltétel nélküli kiadása megszünteti a szűrő hatását.

Ha a SET FILTER paranccsal szűrőfeltételt határozunk meg, akkor az aktuális munkaterületen a műveletek úgy hajtódnak végre, mintha az adatbázis csak a szűrőfeltételnek megfelelő rekordokból állna. Kivételt ez alól az INDEX és REINDEX parancsok — amelyek minden rekordot figyelembe vesznek —, valamint a GOTO, és a RECORD paramétert tartalmazó parancsok képeznek, amelyekkel direkt módon az egyébként „nem látható" rekordok is elérhetők. Az érvényes szűrőfeltétel a DBFILTER() függvénnyel lekérdezhető.

A szűrőfeltétel kiadása után a rekordmutató nem áll automatikusan a feltételnek megfelelő első rekordra, ehhez a GO TOP utasítást kell kiadnunk.

Minden munkaterülethez tartozhat egy szűrő. Ha a szűrőfeltételben szereplő mezők értéke úgy változik meg, hogy az már nem felel meg a feltételnek, a rekord „láthatatlanná" válik.

Példa:

A feldolgozáshoz az adatbázisból csak azok a rekordok szükségesek, amelyeknél az év értéke kisebb 87-nél.

```
USE Sue NEW
SET FILTER TO ev < 87
LIST eset, ev, ertek
SET FILTER TO
```

Helye: CLIPPER.LIB

Lásd még: SET DELETED, DBFILTER()

SET FIXED

A tizedesjegyet tartalmazó számok fix hosszúságú írása

SET FIXED on|OFF|<logikai kifejezés>

ON — a SET DECIMALS által meghatározott számú tizedesjegyet ír ki.

OFF — a tizedesjegyek száma a számítás végeredményétől függ: az értéktelen nullák elhagyásra kerülnek.

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően a tizedesjegyek száma rögzítve van, vagy nincs. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

A SET FIXED pontosan teljesíti a SET DECIMALS által meghatározott számú tizedesjegy kiíratását (ha az nincs megadva, akkor az alapértelmezés szerint két tizedest vesz). Ha SET FIXED OFF van érvényben, akkor a tizedesjegyek száma maximum a SET DECIMALS által megadott érték, vagy ennél kisebb, ha a számítások eredménye kevesebb tizedesjegyet eredményez.

A SET FIXED és a SET DECIMALS csak az eredmények megjelenítésére van hatással, a számítások pontosságára nincs!

Helye: CLIPPER.LIB

Lásd még: SET DECIMALS, EXP(), LOG(), SQRT(), VAL()

SET FORMAT*

Formátumállomány aktivizálása a READ parancs kiadásakor

SET FORMAT TO [<állománynév>[.<kiterjesztés>]]

<állománynév> — lehet formátumállomány (.fmt), program (.prg), vagy eljárás neve.

<kiterjesztés> — a formátumállomány kiterjesztése. Ha nincs megadva, az alapértelmezés .fmt.

SET FORMAT TO magában megadva megszünteti a korábbi FORMAT parancs hatását.

A SET FORMAT egy olyan eljárást definiál, amely a READ parancs kiadásával kerül végrehajtásra. A SET FORMAT által meghatározott eljárást a Clipper fordító DO parancsként értelmezi. Ha ilyen nevű eljárással már találkozott, akkor azt veszi figyelembe; ha nem, akkor a lemezen keresi a megadott nevű állományt. Ha ott sem találja, akkor külső eljárásnak tekinti, amelyet a szerkesztés során kell kapcsolni.

A SET FORMAT a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem javasolt!

Fontos tudnivalók:

- A Clipper egyszerre az összes munkaterület számára csak egy aktív formátumállomány használatát engedi meg.
- A formátumeljárás működése alatt a képernyő törlését a Clipper nem hajtja végre.

- A @...SAY és a @...GET parancsok részét képező utasítások és parancsok a formátumeljárásban is végrehajtásra kerülnek.
- A formátumeljárás egymásból való többszörös hívása nincs megengedve!

Példa:

Az alábbi példában addig folytatódik az adatok bevitele, amíg a felhasználó az ESCAPE billentyűvel azt meg nem szakítja:

```
USE Sue NEW
SET FORMAT TO Sueadat
DO WHILE LASTKEY() != 27
   APPEND BLANK
   READ
ENDDO
RETURN

PROCEDURE Sueadat
   @ 12, 12 SAY "Dátum: " GET datum
   @ 13, 12 SAY "Esemény: " GET eset
RETURN
```

Helye: CLIPPER.LIB

Lásd még: @...GET, @...SAY, PROCEDURE, READ

SET FUNCTION

Értékadás a funkcióbillentyűk számára

SET FUNCTION <funkcióbill_szám> TO <karakterlánc>

<funkcióbill_szám> — annak a funkcióbillentyűnek a sorszáma, amelyhez értéket kívánunk hozzárendelni.

<karakterlánc> — a funkcióbillentyűhöz rendelt érték.

A SET FUNCTION az 1 és 40 közé eső sorszámmal ellátott funkcióbillentyűkhöz biztosít értékadási lehetőséget. Hatására az illető funkcióbillentyű lenyomása után az abban szereplő érték a billentyűzetpufferbe kerül. A <karakterlánc> tartalmazhat vezérlőkaraktereket (például Ctrl-C, Ctrl-S) is. A ; (pontosvessző) karakter a Return billentyűt jelöli.

A funkcióbillentyűk sorszámozása:

1 -10	F1—F10
11-20	Shift-F1—Shift-F10
21-30	Ctrl-F1—Ctrl-F10
31-40	Alt-F1—Alt-F10

Fontos! A Clipper 5.0 verzióban — az előző verzióktól eltérően — a SET FUNCTION felülírja egy korábbi SET KEY által definiált funkcióbillentyű értéket, illetve ugyanez fordítva is igaz. Ez inkompatibilitást jelent a korábbi verziókhoz képest!

Helye: CLIPPER.LIB

Lásd még: KEYBOARD, SET KEY

SET INDEX

Indexállomány(ok) megnyitása az aktuális munkaterületen

SET INDEX TO [<indexáll_lista>]

<indexáll_lista> — az egyidejűleg megnyitásra kerülő indexállományok (.ndx, vagy .ntx) vesszővel elválasztott listája, amelyek száma maximum 15 lehet. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. A változókkal megadott állományneveknél a "" (üres név), illetve a kiegészítő szóközök figyelmen kívül maradnak. Ha kiterjesztést nem adunk meg, az alapértelmezés .ntx Clipper esetében és .ndx a dBase III Plus esetében.

A magában megadott SET INDEX TO a munkaterület összes nyitott indexállományát lezárja.

A SET INDEX kiadásakor az összes eddig megnyitott indexállomány lezárásra kerül, és a paraméterlistában felsorolt indexállomány(oka)t nyitja meg. Ha egyszerre egynél több indexállományt nyitunk meg, az első lesz az ellenőrző index. A rekordmutató automatikusan az index szerinti első rekordra áll. Módosítás, vagy új adat felvétele minden nyitott indexállományban egyidejűleg megy végbe, beleértve a kulcsmezők módosítását is. Az ellenőrző index váltását (új SET INDEX parancs kiadása helyett) a SET ORDER paranccsal végezhetjük el.

A SET INDEX parancs alkalmazása elsődlegesen a hálózatos alkalmazásoknál javasolt, a USE ... INDEX ... paranccsal történő egyidejű adatbázis és indexállomány megnyitás helyett (lásd példa).

Példák:

Változók alkalmazása az indexállományok nevének közvetlen megadása helyett:

```
ind1 = "nev"
ind2 = "cim"
SET INDEX TO (nev), (cim)
```

Hálózatos alkalmazásnál az adatbázis megnyitása után megvizsgáljuk, hogy az sikeresen fejeződött-e be, és ha igen, csak akkor nyitjuk meg az indexállományokat:

```
USE Sue SHARED
IF !NETERR()
   SET INDEX TO nev, cim
ELSE
   ? "Az állományt másik állomás használja!"
   BREAK
ENDIF
```

Helye: CLIPPER.LIB

Lásd még: CLOSE, INDEX, REINDEX, SET ORDER, USE

SET INTENSITY

A kiemelt színek használatának engedélyezése

SET INTENSITY ON|off|<logikai kifejezés>

ON — engedélyezi mind a <normál>, mind a <kiemelt> színek használatát (alapértelmezés).

OFF — csak a <normál> szín használatát engedélyezi.

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően a <kiemelt> szín használata engedélyezve van, vagy nincs. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

A SET INTENSITY ON állapotban a GET és a PROMPT parancsoknál az aktuális mezőt, illetve menüpontot a SET COLOR parancsban megadott <kiemelt> színnel jeleníti meg. OFF állapotban csak a normál színt használja (így a PROMPT parancsnál, ahol a kurzor rejtve van, nem látható, hogy melyik az aktuális).

A SET INTENSITY parancs nem fejti ki hatását az ACHOICE() és a DBEDIT() függvények esetében.

Helye: CLIPPER.LIB

Lásd még: @...GET, @...PROMPT,@...SAY, SETCOLOR()

SET KEY

Eljárás hozzárendelése billentyűhöz

SET KEY <INKEY_kód> TO [<eljárásnév>]

<INKEY_kód> — a billentyű INKEY kódja.

TO <eljárásnév> — a billentyűhöz hozzárendelt eljárás neve. Ha <eljárásnév> nincs megadva, az előző hozzárendelés törlődik.

A SET KEY parancs — a megadott kódú billentyű lenyomása esetén — egy felfüggesztési (wait) állapotban meghívja a TO paraméter után definiált eljárást. Ilyen felfüggesztési állapotot idéznek elő a billentyűzetről értéket váró parancsok és függvények (ACHOICE(), DBEDIT(), MEMOEDIT(), ACCEPT, INPUT, READ és WAIT — az INKEY() kivételével). A billentyűhöz történő hozzárendelés után automatikusan három érték generálódik: PROCNAME() — a hívó eljárás neve (karakteres); PROCLINE() — a hívó eljárás aktuális sorának száma (numerikus); READVAR() — annak a változónak a neve, amelyet éppen kitölt (karakteres).

Egyidejűleg maximum 32 billentyűhöz rendelhető eljárás. Az F1 funkcióbillentyűhöz a rendszer automatikusan a HELP-et rendeli hozzá.

A SET KEY eljárások megalkotásakor a programozónak figyelemmel kell lenni a különböző helyen történő alkalmazási körülményekre (képernyőtartalom. aktuális munkaterület stb.), és ezek megváltoztatása esetén vissza kell állítani az eredeti állapotot.

Fontos! A Clipper 5.0 verzióban — az előző verzióktól eltérően — a SET FUNCTION felülírja egy korábbi SET KEY által definiált funkcióbillentyű értéket, illetve ugyanez fordítva is igaz. Ez inkompatibilitást jelent a korábbi verziókhoz képest!

Fontos tudnivalók:

- A SET KEY szerinti értékadás felülírja a SET ESCAPE és a SETCANCEL() hatását.
- A SET KEY eljáráson belüli CLEAR paranccsal nem lehet a képernyőt törölni, ha READ-re váró GET értékek vannak érvényben. A képernyő törlésére használjuk a CLEAR SCREEN, vagy a CLS parancsokat.
- A READ parancs hatása SET KEY eljárással a következő befejezési módokkal szüntethető meg:

Parancs	Hatás
CLEAR GETS	A GET értékek elvesznek
BREAK	A GET értékek elvesznek
KEYBOARD Ctrl-W	A GET értékek elmentésre kerülnek
KEYBOARD Esc	A GET értékek elvesznek

Példa:

Bemutatjuk hogyan lehet adatbevitel közben a mezőhöz kapcsolódó információkat nyújtani (az inkey.ch a funkcióbillentyűk konstansokkal való megfeleltetését tartalmazza). A vállalat azonosítószáma mező kitöltéséhez a vállalatok listájából választjuk ki a számunkra szükséges-set

```
#include "Inkey.ch"
//
SET KEY K_F2 TO vallista
USE valltor NEW
USE forgalom NEW
@ 10,10 GET forgalom->vallazon
READ
RETURN


PROCEDURE vallista( procnev, sor, valtozo )
   IF valtozo = "VALLAZON"
SAVE SCREEN
valltor->(DBEDIT(10,10,18,40,("vallnev")))
KEYBOARD CHR(K_CTRL_Y) + valltor->vallazon + CHR(K_HOME)
RESTORE SCREEN
   ELSE
TONE(100, 2)
   ENDIF
RETURN
```

Helye: CLIPPER.LIB, segédállomány: Inkey.ch

Lásd még: KEYBOARD, PROCEDURE, READ, SET, ESCAPE, SET FUNCTION, INKEY(), LASTKEY(), PROCLINE(), PROCNAME(), READVAR(), SETKEY()

Fontos! A Clipper 5.0 verzióban — az előző verzióktól eltérően — a SET FUNCTION felülírja egy korábbi SET KEY által definiált funkcióbillentyű értéket, illetve ugyanez fordítva is igaz. Ez inkompatibilitást jelent a korábbi verziókhoz képest!

Fontos tudnivalók:

- A SET KEY szerinti értékadás felülírja a SET ESCAPE és a SETCANCEL() hatását.
- A SET KEY eljáráson belüli CLEAR paranccsal nem lehet a képernyőt törölni, ha READ-re váró GET értékek vannak érvényben. A képernyő törlésére használjuk a CLEAR SCREEN, vagy a CLS parancsokat.
- A READ parancs hatása SET KEY eljárással a következő befejezési módokkal szüntethető meg:

Parancs	Hatás
CLEAR GETS	A GET értékek elvesznek
BREAK	A GET értékek elvesznek
KEYBOARD Ctrl-W	A GET értékek elmentésre kerülnek
KEYBOARD Esc	A GET értékek elvesznek

Példa:

Bemutatjuk hogyan lehet adatbevitel közben a mezőhöz kapcsolódó információkat nyújtani (az inkey.ch a funkcióbillentyűk konstansokkal való megfeleltetését tartalmazza). A vállalat azonosítószáma mező kitöltéséhez a vállalatok listájából választjuk ki a számunkra szükséges-et

```
#include "Inkey.ch"
//
SET KEY K_F2 TO vallista
USE valltor NEW
USE forgalom NEW
@ 10,10 GET forgalom->vallazon
READ
RETURN


PROCEDURE vallista( procnev, sor, valtozo )
   IF valtozo = "VALLAZON"
SAVE SCREEN
valltor->(DBEDIT(10,10,18,40,("vallnev")))
KEYBOARD CHR(K_CTRL_Y) + valltor->vallazon + CHR(K_HOME)
RESTORE SCREEN
   ELSE
TONE(100, 2)
   ENDIF
RETURN
```

Helye: CLIPPER.LIB, segédállomány: Inkey.ch

Lásd még: KEYBOARD, PROCEDURE, READ, SET, ESCAPE, SET FUNCTION, INKEY(), LASTKEY(), PROCLINE(), PROCNAME(), READVAR(), SETKEY()

SET MARGIN

Margóállítás a nyomtatón

SET MARGIN TO [<numerikus kifejezés>]

<numerikus kifejezés> — pozitív egész szám, amely nyomtatásnál azt határozza meg, hogy a lap bal oldalától hány oszlopnyira lesz a margó. Negatív érték megadása esetén 0-nak veszi.

SET MARGIN TO magában megadva az alapértelmezést — 0 értéket jelenti.

A SET MARGIN parancsban megadott érték érvényes minden nyomtatási műveletnél, valamint a @...SAY nyomtatóra átirányított esetében is. Soremelések után a <numerikus kifejezés>-ben megadott értékkel beljebb kezdődik a sor nyomtatása. A @...SAY szerinti nyomtatásnál a parancsban megadott sor- és oszlopértékek a margó értékével megnövelődnek.

Példa:

A nyomtatásnál 5 oszlopnyi legyen a margó:

```
USE Sue NEW
SET MARGIN TO 5
LIST datum, eset TO PRINTER
SET MARGIN TO
```

Helye: CLIPPER.LIB

Lásd még: @...SAY, SET DEVICE, SET PRINTER, PCOL()

SET MESSAGE

A PROMPT megjegyzéssorának definiálása

SET MESSAGE TO [<sor> [CENTER|CENTRE]]

<sor> — a megjegyzés elhelyezésének sora.

CENTER|CENTRE — a megjegyzést a <sor> közepére igazítja.

A SET MESSAGE TO 0 ÉS A SET MESSAGE TO érték nélküli megadása törli a megjegyzéssor megjelenítését.

A SET MESSAGE paranccsal lehet meghatározni, hogy a @...PROMPT paranccsal definiált menü egyes menüpontjaihoz tartozó megjegyzések a képernyő melyik sorában jelenjenek meg. Az alapértelmezés 0, ami azzal egyenértékű, hogy nincs megjegyzés fűzve az egyes menüpontokhoz. Ha a CENTER paraméter van megadva, és a megjegyzés hosszabb a rendelkezésre álló helynél, akkor a 0. oszlopban kezdődik a kiírása.

Példa:

A megjegyzéssor használatának tipikus alkalmazása:

```
SET MESSAGE TO 23 CENTER
SET WRAP ON
@ 5,5 PROMPT "Elsö  " MESSAGE "Elsö menüpont"
@ 6,5 PROMPT "Másod " MESSAGE "Második menüpont"
```

```
MENU TO menuval
//
IF menuval == 0
   EXIT
ELSEIF menuval == 1
   proc1()
ELSEIF menuval == 2
   proc2()
ENDIF
```

Helye: CLIPPER.LIB

Lásd még: @...PROMPT, MENU TO, SET WRAP

SET ORDER

Az ellenőrző index váltása

SET ORDER TO [<numerikus kifejezés>]

<numerikus kifejezés> — az új ellenőrző index sorszáma, amely meghatározza, hogy az aktuális munkaterületen megnyitott indexállományok közül melyik szerint történjenek a további műveletek. Értéke 0-15 közötti szám lehet. Ha a <numerikus kifejezés>-nek nem adunk meg semmilyen értéket, az alapértelmezés 0.

A SET ORDER TO paranccsal tudjuk meghatározni, hogy az adatbázis egyidejüleg megnyitott indexállományai közül a keresési stb. műveleteket melyik index szerinti sorrendben kívánjuk végrehajtani. Függetlenül az ellenőrző indextől, az állományon végzett módosítások és egyéb műveletek mindegyik nyitott indexállományban automatikusan és egyidejűleg átvezetésre kerülnek. Az ellenőrző index váltásakor a rekordmutató ugyanazon a rekordon marad.

A SET ORDER TO 0, illetve a SET ORDER TO parancs hatására az adatbázis az eredeti (rekordmutató szerinti) fizikai sorrendben lesz elérhető, ugyanakkor az összes indexállomány nyitva marad.

Az ellenőrző (aktuális) index lekérdezését az INDEXORD() függvénnyel végezhetjük.

Példa:

A SET ORDER változásai szerinti indexállományok illusztrálása:

```
USE Sue NEW
INDEX ON nev TO Suenev
INDEX ON varos+utca TO Lakas
SET INDEX TO Suenev,Lakas
//
SET ORDER TO 2
? INDEXKEY(INDEXORD())      // Eredmény: varos+utca
//
SET ORDER TO 0
? INDEXKEY(INDEXORD())      // "Eredmény: üres érték"
//
SET ORDER TO 1
```

```
? INDEXKEY(INDEXORD())        // Eredmény: nev
```

Helye: CLIPPER.LIB

Lásd még: INDEX, SEEK, SET INDEX, USE, INDEXORD()

SET PATH

Az állományok keresési útvonalának meghatározása

SET PATH TO [<elérési útvonal>]

<elérési útvonal> — azonosítja azt az elérési útvonalat, amelyen a Clipper az állományokat keresi, ha az aktuális könyvtárban nem találja. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. Az elérési útvonalat alkotó könyvtárneveket vesszővel, vagy pontosvesszővel kell egymástól elválasztani, így ez az egyetlen parancs, amelyben a sor elválasztása a pontosvesszővel nem lehetséges.

A SET PATH parancs segítségével adhatjuk meg azokat a könyvtárakat, amelyben a program által használt állományok az aktuális (DEFAULT) könyvtáron kívül előfordulhatnak. Figyeljünk azonban arra, hogy néhány memória- és alacsonyszintű állományfüggvény nem veszi figyelembe a Clipper DEFAULT és PATH parancsát!

Az elérési útvonal tartalmazhat meghajtóazonosítót (kettősponttal), valamint ezt követően könyvtárneveket a \ jellel kezdve.

Egy állomány keresése során a Clipper először az aktuális könyvtárban keres, ami a SET DEFAULT szerint meghatározott, vagy a DOS szerinti, ha az előbbi nincs definiálva. Ha az állományt nem találja, akkor az elérési útvonalon felsorolt sorrendben haladva minden könyvtárban megnézi, ott van-e a szükséges állomány.

A SET PATH TO magában megadva azt jelenti, hogy a keresés csak az aktuális könyvtárban következik be.

Példák:

Egy tipikus PATH utasítás:

```
SET PATH TO A:\SUSAN;B:\DEDI
```

Ha az elérési útvonalat a felhasználói program módosítása nélkül DOS-ból kívánjuk a későbbiekben változtatni, azt a GETENV() függvény felhasználásával az alábbi módon tehetjük meg:

```
C>SET CLIP_PATH=C:\DEDI\SUE,C:\DEDI\LISE      //DOS-ban adva meg
```

Az így definiált elérési útvonalat például az alábbi módon használhatjuk programunkban:

```
SET PATH TO (GETENV("CLIP_PATH"))
```

Helye: CLIPPER.LIB

Lásd még: SET DEFAULT, CURDIR()

SET PRINTER

A megjelenítés átirányítása nyomtatóra vagy állományba

SET PRINTER on|OFF|<logikai kifeejzés>
SET PRINTER TO [<eszköznév>|<állománynév>]

ON — A megjelenítés a nyomtatóra történik.

OFF — A nyomtatón nincs megjelenítés (alapértelmezés).

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően a nyomtatóra írás engedélyezve van, vagy nincs. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

TO <eszköznév> — annak az eszköznek a neve, amelyre az összes output művelet irányul. Megadhatjuk közvetlenül az eszköznevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. Az eszköz egyaránt lehet lokális vagy hálózati. Ha a megadott névvel nem létezik eszköz, akkor ezzel a névvel egy állomány kerül létrehozásra.

TO <állománynév> — az output állomány neve. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. Ha a kiterjesztést nem adjuk meg, .prn az alapértelmezés.

Ha a SET PRINTER TO magában áll, akkor a korábban definiált eszköz, illetve állomány lezárásra kerül.

A SET PRINTER on|OFF forma azt határozza meg, hogy a megjelenítés a képernyőn kívül más eszközre is szükséges-e. A konzolparancsok a ?|?? parancsok kivételével rendelkeznek TO PRINTER paraméterrel, amely szintén a közvetlen nyomtatóra írást teszi lehetővé. Ha a nyomtatóra írás közben a képernyőn való egyidejű megjelenítést le akarjuk tíltani, akkor a SET CONSOLE OFF parancsot kell kiadni. A @...SAY parancsra a SET PRINTER ON nincs hatással, nyomtatásához a SET DEVICE TO PRINTER formát kell alkalmazni.

A SET PRINTER TO ezzel szemben minden parancs és függvény (beleértve a @...SAY parancsot is, ha a SET DEVICE TO PRINTER meg lett adva) esetében kifejti hatását. Az output irányítása megválasztható: mehet a nyomtatóra vagy lemezállományba. Ha az irányítás eszköz szerinti, akkor a megenegdett eszköznevek: LPT1, LPT2, LPT3 (az összes párhuzamos port), COM1, COM2 (soros portok) és PRN. Az alapértelmezés PRN.

Ha az irányítás állomány szerinti, az az aktuális DEFAULT könyvtárba íródik. Ha ilyen nevű állomány ott már létezik, az figyelmeztető üzenet nélkül felülíródik. Mindaddig történik ebbe az állományba az írás, amig SET PRINTER TO utasítással le nem zárjuk azt.

Fontos tudnivalók:

- A Clipper nem tudja értelmezni a //SPOOLER vagy //CAPTURE parancsokat, helyettük SPOOLER.PRN, illetve CAPTURE.PRN néven állományokat hoz létre, a //-t figyelmen kivül hagyja.
- Ha a nyomtatás átirányítása egy állományba történik, az állományvége-jel (end-of-file: CHR(26)) nem kerül felírásra a lezárás során. Ezért az állomány közvetlen lezárása előtt a programban adjunk ki egy ?? CHR(26) parancsot.

Példák:

A példában a megjelenítés átirányításával egyidejűleg letíltjuk a képernyőt:

```
USE Sue NEW
SET PRINTER ON
SET CONSOLE OFF
DO WHILE !EOF()
   ? Sue->neve, Sue->tel
   SKIP
ENDDO
EJECT
SET PRINTER OFF
SET CONSOLE ON
CLOSE
RETURN
```

A nyomtatás átirányítása az LPT1 eszközre:

```
SET PRINTER TO LPT1
   <nyomtatási utasítások">
SET PRINTER TO
```

A megjelenítés átirányítása egy állományba:

```
SET PRINTER TO nyom.txt
SET DEVICE TO PRINTER
SET PRINTER ON
//
@ 0,0 SAY "Ez a nyom.txt állományba megy!"
? "Minden rendben"
//
SET DEVICE TO SCREEN
SET PRINTER OFF
SET PRINTER TO
```

Helye: CLIPPER.LIB

Lásd még: @...SAY, EJECT, SET CONSOLE, SET DEVICE

SET PROCEDURE*

Eljárás vagy függvény befordítása a tárgymodulba

SET PROCEDURE TO [<eljárásnév>[.<kiterjesztés>]]

<eljárásnév> — annak az eljárásnak vagy függvénynek a neve, amelyet a tárgymodulba be kívánunk fordítani. Tartalmazhatja a meghajtót és elérési útvonalat is.

<kiterjesztés> — az eljárás kiterjesztése. Ha nem adjuk meg, az alapértelmezés .prg.

SET PROCEDURE TO önmagában, paraméter nélkül, figyelmen kívül marad.

A SET PROCEDURE parancs hatására a megadott eljárás vagy felhasználó által definiált függvény a fordítás során bekerül a készülő tárgymodulba (.OBJ állományba).

A SET PROCEDURE a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem javasolt!

Helye: CLIPPER.LIB

Lásd még: DO, FUNCTION, PROCEDURE, RETURN, #include

SET RELATION

Két munkaterület kulcsérték vagy rekordsorszám szerinti összekapcsolása

SET RELATION TO [<kulcskif1>|<rek_sorszám> INTO <fiu_mter>]
[,<kulcskif2>|<rek_sorszám2> INTO <fiu_mter2>...] [ADDITIVE]

TO <kulcskif> — egy olyan kifejezés, ami szerint a fiú munkaterületen a SEEK-kel történő keresés a szülő adatbázis rekordmutató változásával egyidejűleg végrehajtható. Ehhez a fiú munkaterületen levő adatbázisnak legalább egy nyitott indexállománnyal kell rendelkeznie!

TO <rek_sorszám> — egy olyan kifejezés, amelyet a fiú-munkaterületen egy GOTO utasítás használ fel a szülő adatbázis rekordmutató változásával egyidejűleg. Ha a <rek_sorszám>-ként a RECNO() függvényt adjuk meg, akkor a fiú munkaterületen automatikusan megkeresésre kerül az ugyanolyan sorszámú rekord, mint amelyen a szülő munkaterületen áll. Ebben az esetben a fiú munkaterületen nem szabad nyitott indexállománynak lennie.

INTO <fiu_mter> — a fiú munkaterület azonosítása. Megadhatjuk közvetlenül a munkaterület nevét, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

ADDITIVE — a paranccsal definiált kapcsolatot az eddig már felépített kapcsolatokhoz fűzi. Ha nincs megadva, akkor törlődnek az eddigi kapcsolatok, és csak az itt definiált(ak) az érvényes(ek).

Az argumentum nélküli SET RELATION TO megszünteti a munkaterület összes érvényben levő relációs kapcsolatát.

A SET RELATION relációs kapcsolatot teremt a szülő munkaterület és egy vagy több fiú munkaterület között. A kapcsolat létrehozásához kulcskifejezést, rekordsorszámot, vagy numerikus kifejezést használ fel. Egy szülő munkaterület egyszerre legfeljebb nyolc fiú munkaterülettel állhat relációs kapcsolatban. A fiú munkaterületen a rekordmutató a relációs feltételnek megfelelően változik. Ha a feltételnek megfelelő rekord a fiú munkaterületen nem található, akkor a rekordmutató értéke LASTREC() + 1, AZ EOF() értéke igaz (.T.), a FOUND() értéke hamis (.F.).

Fontos tudnivalók:

- A szülő munkaterület nem állhat relációs kapcsolatban önmagával sem direkt, sem indirekt módon!
- A SET RELATION nem veszi figyelembe a SET SOFTSEEK helyzetét, és mindig úgy viselkedik, mintha az OFF állásban lenne. Ezért, ha a kapcsolat feltételének megfelelő rekord a fiú adatbázisban nem található, akkor a rekordmutató értéke mindig LASTREC() + 1.
- Ha két adatbázis ugyanolyan rekordsorszámokkal rendelkezik, akkor használjuk a rekordsorszám szerinti kapcsolatteremtést a RECNO() függvénnyel! Ebben az

esetben azonban vigyázzunk arra, hogy ne legyen egyetlen nyitott indexállomány sem érvényben!

Példák:

Három munkaterület többszörös szülő-gyermek relációs kapcsolata:

```
USE Arutor INDEX Arutor NEW
USE Eladok INDEX Elad NEW
USE Termelok NEW
SET RELATION TO termkod INTO Arutor, Elad INTO Eladok
LIST Termel,Eladok->Thelye,Arutor->arukod,Arutor->ara
```

Később egy új relációs kapcsolatot kívánunk az eddigiekhez kapcsolni, ehhez az ADDITION paraméter megadása szükséges:

```
USE Szallitok INDEX Szallitok NEW
SELECT Termelok
SET RELATION TO termkod INTO Szallitok ADDITIVE
```

Helye: CLIPPER.LIB

Lásd még: SET INDEX, SET ORDER, SET SOFTSEEK, DBRELATION(), DBRSELECT()

SET SCOREBOARD

Üzenetek megjelenítése READ és MEMOEDIT() közben

SET SCOREBOARD ON|off|<logikai kifejezés>

ON — Ebben az állásban engedélyezi a READ és a MEMOEDIT() üzeneteinek megjelenését a képernyő 0. sorában.

OFF — nem engedélyezi az üzenetek megjelenését.

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően a képernyőre írás engedélyezve van, vagy nincs. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

A SET SCOREBOARD parancs ellenőrzi a READ és a MEMOEDIT() üzeneteinek megjelenítési lehetőségét. Ha az ON paraméter van érvényben, akkor a képernyő 0. sorában megjelennek a RANGE által felismert hibák, a hibás dátumok, valamint kijelzésre kerül az INSERT állapot. A MEMOEDIT() függvénynél a megszakított kérdésekkel kapcsolatos üzenetek, valamint az INSERT állapot.

Ezeknek az üzeneteknek a letíltása a SET SCOREBOARD OFF paranccsal lehetséges.

Helye: CLIPPER.LIB

Lásd még: @...GET, READ, MEMOEDIT()

SET SOFTSEEK

Legközelebbi rekord keresése

SET SOFTSEEK on|OFF|<logikai kifejezés>

ON — meghatározza, hogy a rekordmutató a keresett rekordnál nagyobb, de ahhoz legközelebb álló rekordra álljon, ha a keresett rekord nem létezik.

OFF — ha a keresett rekord nem létezik, a rekordmutató az állomány végére megy.

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően a legközelebbi rekord keresése engedélyezve van, vagy nincs. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

A SET SOFTSEEK segítségével biztosíthatjuk, hogy minden esetben valamilyen rekordértéket kapjunk vissza a keresés után, akkor is, ha a keresett rekordnak pontosan megfelelőt nem talált az adatbázisban. Ez csak akkor nem teljesül, ha a keresett feltételnél nagyobb kulcsú rekord nem létezik az adatbázisban.

Ha a SET SOFTSEEK ON, és a SEEK feltételben megadott rekord nem található, a rekordmutató a keresett kulcsnál nagyobb, de ahhoz legközelebb álló rekordra lép. A SET FILTER és a SET DELETED parancsok által kiszűrt rekordokat nem veszi figyelembe. Ha a keresett rekordnál nagyobb rekord nem létezik az adatbázisban, akkor a rekordmutató értéke LASTREC()+1, az EOF() értéke igaz, (.T.), a FOUND () értéke hamis (.F.). A FOUND() értéke a legközelebbi rekord megtalálása esetén mindig hamis (.F.) (csak akkor igaz (.T.), ha pontosan a keresett rekordot találta meg).

Ha a SET SOFTSEEK OFF, és a SEEK nem találta a keresett rekordot, a rekordmutató értéke LASTREC()+1, az EOF() igaz (.T.) értéket ad vissza, a FOUND() értéke hamis (.F.).

Példa:

A SEEK szerinti közelítő keresés SET SOFTSEEK segítségével:

```
SET SOFTSEEK ON
USE Sue INDEX Sue NEW
ACCEPT "Add meg a dátumot: " TO datum
SEEK datum
DO CASE
CASE FIELD->Sue = datum
   ? "Pontos találat: ", FOUND(), EOF(), FIELD->Sue
CASE !EOF()
   ? "Közelít találat! ", FOUND(), EOF(), FIELD->Sue
OTHERWISE
   ? "Nincs találat0 ", FOUND(), EOF(), FIELD->Sue
ENDCASE
```

Helye: CLIPPER.LIB

Lásd még: SEEK, SET INDEX, SET ORDER, SET RELATION, FOUND()

SET TYPEHEAD

A Clipper billentyűzetpuffer méretének megadása

SET TYPEHEAD TO <pufferméret>

<pufferméret> — a billentyűzetpuffer mérete, amely 0-tól 4096-ig terjedhet. Az alapértelmezés géptől függő, de minimum 16.

A SET TYPEHEAD a *Clipper* billentyűzetpufferének méretét határozza meg. A parancs kiadásakor először törlődik a billentyűzetpuffer, majd beállításra kerül a megadott pufferméret.

SET TYPEHEAD TO 0 megadásával programunk rendkivüli módon felgyorsítható, mivel a billentyűzetpuffer szerepét a BIOS puffer veszi át. A NEXTKEY() függvény kiolvassa a BIOS puffer tartalmát, és annak INKEY() értékét használhatjuk a továbbiakban (illetve ez az érték 0, ha a BIOS puffer üres).

Helye: CLIPPER.LIB

Lásd még: CLEAR TYPEHEAD, KEYBOARD, ALTD(), INKEY(), NEXTKEY()

SET UNIQUE*

Indexállomány egyedi kulcsértékek szerinti használata

SET UNIQUE on|OFF|<logikai kifejezés>

ON — az indexállományt az egyedi kulcsértékek szerint használjuk.

OFF — a teljes indexállományt használjuk (alapértelmezés).

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően használjuk az eredeti (teljes) indexállományt, vagy csak az egyedi kulcsértékek szerintit. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

SET UNIQUE ON esetén az új indexállomány az azonos kulcsú rekordok közül csak az elsőt tartalmazza. Hatása megegyezik az INDEX ... UNIQUE paranccsal.

Ha azonos kulcsértékkel két vagy több rekord rendelkezik, és ezek közül az indexállomány részét alkotó rekordot töröljük, vagy kulcsértékét módosítjuk, csak az első rekord törlődik, illetve csak ennek kulcsértéke módosul.

A kulcsérték módosítása esetén:

- ha az új kulccsal már létezik rekord az indexállományban, akkor a módosított rekord kikerül abból;
- a régi kulcsérték szerint nem lesz rekord az indexállományban mindaddig, amíg egy REINDEX, PACK, vagy INDEX ... UNIQUE parancsot ki nem adunk.

A SET UNIQUE a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem javasolt! Helyette inkább az INDEX parancs UNIQUE paraméterét használjuk!

Helye: CLIPPER.LIB

Lásd még: INDEX, PACK, REINDEX, SEEK

SET TYPEHEAD

A Clipper billentyűzetpuffer méretének megadása

SET TYPEHEAD TO <pufferméret>

<pufferméret> — a billentyűzetpuffer mérete, amely 0-tól 4096-ig terjedhet. Az alapértelmezés géptől függő, de minimum 16.

A SET TYPEHEAD a *Clipper* billentyűzetpufferének méretét határozza meg. A parancs kiadásakor először törlődik a billentyűzetpuffer, majd beállításra kerül a megadott pufferméret.

SET TYPEHEAD TO 0 megadásával programunk rendkivüli módon felgyorsítható, mivel a billentyűzetpuffer szerepét a BIOS puffer veszi át. A NEXTKEY() függvény kiolvassa a BIOS puffer tartalmát, és annak INKEY() értékét használhatjuk a továbbiakban (illetve ez az érték 0, ha a BIOS puffer üres).

Helye: CLIPPER.LIB

Lásd még: CLEAR TYPEHEAD, KEYBOARD, ALTD(), INKEY(), NEXTKEY()

SET UNIQUE*

Indexállomány egyedi kulcsértékek szerinti használata

SET UNIQUE on|OFF|<logikai kifejezés>

ON — az indexállományt az egyedi kulcsértékek szerint használjuk.

OFF — a teljes indexállományt használjuk (alapértelmezés).

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően használjuk az eredeti (teljes) indexállományt, vagy csak az egyedi kulcsértékek szerintit. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

SET UNIQUE ON esetén az új indexállomány az azonos kulcsú rekordok közül csak az elsőt tartalmazza. Hatása megegyezik az INDEX ... UNIQUE paranccsal.

Ha azonos kulcsértékkel két vagy több rekord rendelkezik, és ezek közül az indexállomány részét alkotó rekordot töröljük, vagy kulcsértékét módosítjuk, csak az első rekord törlődik, illetve csak ennek kulcsértéke módosul.

A kulcsérték módosítása esetén:

- ha az új kulccsal már létezik rekord az indexállományban, akkor a módosított rekord kikerül abból;
- a régi kulcsérték szerint nem lesz rekord az indexállományban mindaddig, amíg egy REINDEX, PACK, vagy INDEX ... UNIQUE parancsot ki nem adunk.

A SET UNIQUE a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem javasolt! Helyette inkább az INDEX parancs UNIQUE paraméterét használjuk!

Helye: CLIPPER.LIB

Lásd még: INDEX, PACK, REINDEX, SEEK

SET WRAP

Menüválasztás végtelenítése

SET WRAP on | OFF | <logikai kifejezés>

ON — a menüpontok közül történő kiválasztás végtelenítése (az utolsó pont után az elsőre, illetve az elsőről az utolsóra ugrik).

OFF — kikapcsolja a végtelenítés lehetőségét (alapértelmezés).

<logikai kifejezés> — zárójelben kell megadni azt a logikai kifejezést, amelynek eredményétől függően a menükiválasztás végtelenítése engedélyezve van, vagy nincs. Ha a kifejezés kiértékelés utáni értéke igaz (.T.), az az ON állapottal, ha hamis (.F.), az az OFF állapottal egyenértékű.

A SET WRAP ON hatására a @...PROMPT paranccsal definiált menüpontok között a mozgás nemcsak fel-le lehetséges, hanem az utolsó menüponton állva a jobbra, vagy a lefelé mutató nyílat megnyomva az első menüpontra; első menüponton állva és a balra vagy a felfelé mutató nyílat megnyomva az utolsó menüpontra tudunk közvetlenül navigálni.

Ha a SET WRAP OFF, akkor az előzőkben leírt esetekben semmi nem történik.

Helye: CLIPPER.LIB

Lásd még: @...PROMPT, MENU TO, SET MESSAGE

SKIP

A rekordmutató új pozícióba mozgatása

SKIP [<numerikus kifejezés>] [ALIAS <aliasnév> | <munkaterület>]

<numerikus kifejezés> — azt határozza meg, hogy az aktuális rekordtól kezdve milyen mértékben változtassa a rekordmutató értékét. Megadható pozitív, negatív és 0 érték is. A pozitív érték előre, a negatív hátra mozgatja azt.

ALIAS <aliasnév>|<munkaterület> — ha a rerkordmutató léptetését nem az aktuális munkaterületen kívánjuk végezni, akkor itt adhatjuk meg a szükséges munkaterület azonosító nevét (karakteres kifejezésként), vagy a munkaterület sorszámát (numerikus kifejezésként).

SKIP — önmagában megadva a rekordmutató értékét eggyel növeli.

A SKIP az aktuális munkaterületen — az esetleges szűrőfeltétel figyelembevételével — mozgatja az aktuális helyzethez képest a rekordmutatót. Ha az ALIAS kulcsszót megadtuk, akkor a rekordmutató mozgatása egy másik munkaterület adatbázisában lehetséges.

Nyitott indexállomány esetén a léptetés az indexállomány szerinti sorrendben történik, egyébként a rekordok fizikai elhelyezkedésének sorrendjében. Ha az előreléptetés során túlhaladnánk az állományvége (EOF) pozíción, a rekordmutató a LASTREC()+1, az EOF() az igaz (.T.) értéket veszi fel. Ha a visszafelé léptetés során az állománykezdet (BOF) pozíción lépnénk át, akkor a rekordmutató az első rekordra mutat, a BOF() értéke igaz (.T.).

Hálózatban minden léptető utasítás (beleértve a SKIP-et is) megváltoztathatja a közös használatra megnyitott aktuális adatbázis láthatóságát más felhasználó számára, ha RLOCK() van

érvényben. Az állománypuffer tartalmának azonnali lemezre írását érhetjük el a SKIP 0 paranccsal. Ha a léptetés egy FLOCK()-kal zárolt állományon történik, akkor más felhasználó csak akkor láthatja az állományt, ha a zárolást feloldjuk, egy COMMIT parancsot adunk ki, vagy az állományt lezárjuk.

Példák:

A SKIP parancs különböző értékekkel történő kiadása, és ezeknek hatása a rekordmutató helyzetére:

```
USE Sue NEW
SKIP
? RECNO()       // Eredmény: 2
SKIP 17
? RECNO()       // Eredmény: 19
SKIP -5
? RECNO()       // Eredmény: 14
```

A következő példában a rekordmutatót egy másik munkaterületen mozgatjuk:

```
USE Sue NEW
USE Dedi NEW
SKIP ALIAS Sue
```

Egy gyakori alkalmazás: az adatbázis tartalmának kinyomtatása a rekordmutató ciklusban történő egyenkénti léptetésével:

```
LOCAL sor := 99
USE Sue NEW
SET PRINTER ON
DO WHILE !EOF()
   IF sor > 60
      EJECT
      sor := 1
   ENDIF
   ? neve, cime, tel, fax
   sor++
   SKIP
ENDDO
SET PRINTER OFF
CLOSE Sue
```

Helye: CLIPPER.LIB

Lásd még: COMMIT, GO, LOCATE, SEEK, BOF(), EOF(), RECNO()

SORT

Az adatbázis megadott rendezettségben történő másolása

SORT TO <új adatbázis> ON <mező1> [/[A | D][C]]
[,<mező2> [/[A | D][C]]...]
[<hatókör>] [WHILE <feltétel1>] [FOR<feltétel2>]

TO <új adatbázis> — annak az adatbázisnak a neve, amely a rendezett állományt fogja tartalmazni. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. Ha kiterjesztést nem adunk meg, az alapértelmezés .dbf.

ON <mezőx> — a rendezettséget meghatározó mezőnevek. Kötelezően az aktuális adatbázisban szereplő mezőnevek közül kell kikerülniük!

/[A|D][C] — a rendezés módját meghatározó paraméterek:

- /A — növekvő sorrend;
- /D — csökkenő sorrend;
- /C — a kis- és nagybetűk között nem tesz különbséget (az angol ábécére igaz — magyar ékezetes karakterek sorbarendezéséhez saját programot kell készítenünk).

Ha nem adunk meg paramétert, az alapértelmezés a /A (növekvő sorrend).

<hatókör> — az aktuális adatbázisnak a rendezésben résztvevő része. Alapértelmezés: ALL (összes rekord).

WHILE <feltétel1> — a rekordok sorbarendezése mindaddig folyik, amíg a feltétel igaz.

FOR <feltétel2> — a megadott hatókörön belül is csak azok a rekordok vesznek részt a rendezésben, amelyekre a feltétel igaz.

A SORT parancs hatására egy új — a megadott szempontok szerint rendezett — adatbázist hozunk létre, amelynek szerkezete az eredetivel azonos, tartalma vagy a teljes eredeti adatbázis, vagy annak a hatókörrel, illetve a feltételekkel szűkített része. A rendezés az ASCII kódok szerint megy végbe (kivéve ha /C paramétert adunk meg, ebben az esetben a kis- és nagybetűk között nem tesz különbséget). A numerikus mezők rendezése számsorrendben, a dátummezőké kronológiai sorrendben történik, a logikai mezők esetében az igaz (.T.) a nagyobb érték. A memo mezők rendezése nem lehetséges.

A rendezés során a SORT egy Clipsort.tmp nevű ideiglenes állományt hoz létre, amelynek mérete az eredeti adatbázisénál negyobb is lehet.

Hálózatban a rendezésre kerülő állományt FLOCK()-kal, vagy a USE ... EXCLUSIVE megnyitási móddal zárolni kell! A létrejövő állomány automatikusan kizárólagos használatra nyílik meg.

Fontos tudnivalók:

Ha SET DELETED OFF helyzetben van, a SORT a logikailag törölt rekordokat is rendezi és másolja. Ezek a rekordok megtartják logikailag törölt (deleted) állapotukat. DELETED ON esetén a logikailag törölt rekordok nem kerülnek át az új adatbázisba.

A SET EXACT nem befolyásolja a rendezést.

A SET FILTER-rel beállított szűrőfeltételek érvényesek a rendezés során: az új állományba csak a látható" rekordok kerülnek.

Példa:

Egy címlistában név szerint sorbarendezzük a megadott irányítószámmal rendelkezőket, majd kinyomtatjuk ezeket:

```
USE Cimlista NEW
SEEK "1023"
SORT ON elonev, utonev TO Meghivo WHILE irsz = "1023"
USE Meghivo NEW
REPORT FORM Cimzes TO PRINTER
```

Helye: CLIPPER.LIB

Lásd még: INDEX, USE, ASORT(), FLOCK()

STORE*

Értékadás egy vagy több memóriaváltozó számára

STORE <kifejezés> TO <memóriaváltozó-lista>
<memóriaváltozó> = <kifejezés>
<memóriaváltozó> := [<memóriaváltozó2> := ...] <kifejezés>

<kifejezés> — a megadott változó számára átadásra kerülő tetszőleges adattípusú érték.

TO <memóriaváltozó-lista> — egy vagy több *local* , *static* , *public* , *private* , vagy *adatbázismező*-változó neve. Ha a megadott nevű változó nem létezik, vagy az adott helyen nem látható", akkor *private* változóként létrehozásra kerül, és megkapja a <kifejezés> értékét.

A STORE parancs az (=) és a (:=) operátorokkal azonos eredményre vezet. Ha azonos nevű memóriaváltozó és adatbázismező is létezik, akkor a deklarált memóriaváltozónak adja át az értéket. Ha a deklarációtól függetleníteni kívánjuk magunkat, akkor a változó neve előtt a következő előtagok valamelyikét adjuk meg:

- más munkaterületen megnyitott adatbázis mezőneve előtt a munkaterület nevét (például Sue->);
- private vagy public memóriaváltozó esetén: MEMVAR->;
- az aktuális munkaterület adatbázismezője esetén: FIELD->.

A STORE a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem javasolt! A Clipper 5.0-tól kezdve több, ennél hatékonyabb értékadási lehetőség is rendelkezésünkre áll. Ezek közé tartozik az in-line értékadás (:=), a növelés (++), a csökkentés (—), valamint az összetett értékadó operátorok: +=, -=, *=, /=.

A STORE paranccsal csak egyedi tömbelem értékadása végezhető el. A tömb összes elemének feltöltése az AFILL() függvénnyel lehetséges!

Ha egy memo-mezőt egy változóértékkel töltünk fel, a változó típusától függetlenül a memomező értéke a változótartalom karakteres értéke lesz!

A SET FILTER-rel beállított szűrőfeltételek érvényesek a rendezés során: az új állományba csak a látható" rekordok kerülnek.

Példa:

Egy címlistában név szerint sorbarendezzük a megadott irányítószámmal rendelkezőket, majd kinyomtatjuk ezeket:

```
USE Cimlista NEW
SEEK "1023"
SORT ON elonev, utonev TO Meghivo WHILE irsz = "1023"
USE Meghivo NEW
REPORT FORM Cimzes TO PRINTER
```

Helye: CLIPPER.LIB

Lásd még: INDEX, USE, ASORT(), FLOCK()

STORE*

Értékadás egy vagy több memóriaváltozó számára

STORE <kifejezés> TO <memóriaváltozó-lista>
<memóriaváltozó> = <kifejezés>
<memóriaváltozó> := [<memóriaváltozó2> := ...] <kifejezés>

<kifejezés> — a megadott változó számára átadásra kerülő tetszőleges adattípusú érték.

TO <memóriaváltozó-lista> — egy vagy több *local* , *static* , *public* , *private* , vagy *adatbázismező*-változó neve. Ha a megadott nevű változó nem létezik, vagy az adott helyen nem látható", akkor *private* változóként létrehozásra kerül, és megkapja a <kifejezés> értékét.

A STORE parancs az (=) és a (:=) operátorokkal azonos eredményre vezet. Ha azonos nevű memóriaváltozó és adatbázismező is létezik, akkor a deklarált memóriaváltozónak adja át az értéket. Ha a deklarációtól függetleníteni kívánjuk magunkat, akkor a változó neve előtt a következő előtagok valamelyikét adjuk meg:

- más munkaterületen megnyitott adatbázis mezőneve előtt a munkaterület nevét (például Sue->);
- private vagy public memóriaváltozó esetén: MEMVAR->;
- az aktuális munkaterület adatbázismezője esetén: FIELD->.

A STORE a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem javasolt! A Clipper 5.0-tól kezdve több, ennél hatékonyabb értékadási lehetőség is rendelkezésünkre áll. Ezek közé tartozik az in-line értékadás (:=), a növelés (++), a csökkentés (—), valamint az összetett értékadó operátorok: +=, -=, *=, /=.

A STORE paranccsal csak egyedi tömbelem értékadása végezhető el. A tömb összes elemének feltöltése az AFILL() függvénnyel lehetséges!

Ha egy memo-mezőt egy változóértékkel töltünk fel, a változó típusától függetlenül a memomező értéke a változótartalom karakteres értéke lesz!

Példák:

Értékadás korábban nem deklarált változók számára:

```
STORE "ertek" TO valt1, valt2, valt3
valt1 = "ertek2"
valt2 = MEMVAR->valt1
```

A többszörös értékadásra rendlkezésre álló két teljesen egyenértékű lehetőség:

```
STORE "ertek" TO valt1, valt2, valt3
valt1 := valt2 := valt3 := "ertek"
```

Értékadás egy adatbázismező számára, a FIELD előtaggal, illetve a munkaterület alias nevével:

```
USE Sue NEW
FIELD->ertek = 1014
Sue->ertek = 1014
```

Helye: CLIPPER.LIB

Lásd még: PRIVATE, PUBLIC, RELEASE, REPLACE, RESTORE, SAVE

SUM

Numerikus kifejezések összegzése

SUM <kifejezéslista> TO <változólista>
[<hatókör>] [WHILE <feltétel1>] [FOR<feltétel2>]

<kifejezéslista> — az összegzésre kerülő numerikus értékek listája. Lehet egy adatbázismező, memóriaváltozó, vagy tetszőleges numerikus kifejezés.

<változólista> — azoknak a változóknak a listája, amelyekbe az összegek elhelyezésre kerülnek. A korábban még nem deklarált változók *private* változóként létrehozásra kerülnek. A <változólista> elemeinek száma meg kell egyezzen a <kifejezéslista> elemeinek számával.

<hatókör> — az adatbázisnak az összegzésben résztvevő része. Alapértelmezés: ALL (összes rekord).

WHILE <feltétel1> — az összegzés mindaddig folyik, amíg a feltétel értéke igaz.

FOR <feltétel2> — az összegzésben a hatókörön belül is csak a feltételnek megfelelő rekordok vesznek részt.

A SUM parancs segítségével az aktuális munkaterületen megnyitott adatbázis megadott numerikus mezői szerinti összegzés hajtható végre, és az összeg(ek) egy változónak adhatók át. Ez lehet: másik adatbázismező, local, private, public vagy static típusú változó.

A SUM paranccsal tetszőleges numerikus kifejezések összegzése is végrehajtható.

Példa:

A SUM parancs gyakori használati módja:

```
LOCAL osszfiz, osszlev
USE Fizlis NEW
SUM fiz, fiz * .10 TO osszfiz, osszlev
```

```
//
? osszfiz       // 198600.00
? osszlev       // 19860.00
```

Helye: CLIPPER.LIB

Lásd még: AVERAGE, TOTAL, DBEVAL()

TEXT*

Szövegblokk megjelenítése

TEXT [TO PRINTER] [TO FILE <állománynév>]
** <szöveg>...**
ENDTEXT

<szöveg> — a megjelenítendő karaktersorozat. A megjelenítés pontosan az eredetinek megfelelő formában történik.

TO PRINTER — a megjelenítés a nyomtatóra is.

TO FILE <állománynév> — a megjelenítés a megadott állományba is történik. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. Ha kiterjesztést nem adunk meg, az alapértelmezés: .txt.

A TEXT ... ENDTEXT a két kulcsszó közé foglalt szövegblokkot megjeleníti a képernyőn, opcionálisan a nyomtatón és/vagy egy állományban. Ahhoz, hogy a képernyős megjelenítést letíltsuk a nyomtatás, vagy a lemezre írás ideje alatt, a TEXT előtt előzetesen a SET CONSOLE OFF parancsot kell kiadni.

A megjelenítés pontosan a létrehozásnak megfelelő lesz. A kemény soremelés új sorként, a lágy soremelés CHR(141) kódként jelenik meg. Ha a szövegblokkon belül makróváltozó található, annak értéke a megjelenítés során kifejtésre kerül.

A TEXT ... ENDTEXT a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem javasolt! A Clipper ennél jobb megoldásokat is tartalmaz: például a MEMOLINE() és az MLCOUNT() függvények kombinációjával a megjelenítés meghatározott hosszúságú sorokba szóátvitellel történik. A ? vagy a @...SAY a MEMOLINE () függvénnyel együtt formázott megjelenítést tesz lehetővé.

Példa:

A TEXT ... ENDTEXT makrós alkalmazása szabványlevél létrehozásához:

```
USE Cimlis NEW
DO WHILE !EOF()
   Formalev()
   SKIP
ENDDO
RETURN
```

```
FUNCTION Formalev
   LOCAL datum := DTOC(DATE()), cimz := RTRIM(neve)
   TEXT TO PRINTER

   &datum.

   Tisztelt &cimz.,

   "Köszöntöm ..."

   ENDTEXT
   EJECT
   RETURN NIL
```

Helye: CLIPPER.LIB

Lásd még: ?|??, @...SAY, SET CONSOLE, MEMOLINE(), MLCOUNT()

TOTAL

A rekordok kulcsérték szerinti összegzése

TOTAL ON <kulcs> [FIELDS <mezőlista>] TO <új állomány>
[<hatókör>] [WHILE <feltétel1>] [FOR <feltétel2>]

ON <kulcs> — meghatározza azt a rekordcsoportot, amelyekből az összegzés után az <új állomány>-ban egy rekord keletkezik. A pontos összegzés érdekében az aktuális adatbázisnak a <kulcs> szerint indexeltnek vagy rendezettnek kell lennie.

FIELDS <mezőlista> — az összegzendő mezők felsorolása. Ha a FIELDS paramétert nem adjuk meg, összegzés egyetlen mező szerint sem történik, az új állományban az adott kulcsnak megfelelő első rekord értékei szerepelnek.

TO <új állomány> — az összegzett rekordokat tartalmazó új adatbázis neve. Megadhatjuk közvetlenül az állománynevet, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük azt. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. Ha kiterjesztést nem adunk meg, az alapértelmezés .dbf. Hálózatban automatikusan kizárólagos használatra nyílik meg.

<hatókör> — az adatbázis összegzésben résztvevő rekordjainak köre. Az alapértelmezés: ALL (összes rekord).

WHILE <feltétel1> — az összegzés mindaddig történik, amíg a feltétel értéke igaz.

FOR <feltétel2> — az összegzés a hatókörön belül is csak azokra a rekordokra vonatkozik, amelyekre a feltétel teljesül.

A TOTAL parancs segítségével az aktuális adatbázis hatókörrel, vagy feltételekkel kijelölt rekordjai a megadott kulcs szerint összegzésre kerülnek, és ezek az új rekordok egy másik adatbázis rekordjait fogják képezni.

Fontos! A logikailag törölt (deleted) rekordok is összegzésre kerülnek, ha a SET DELETED ON nincs beállítva! Ekkor a rekordok törölt státusza megszünik!

Példa:

Az adatbázist az ellenőrző index kulcsértékének makrós formájával összegezzük:

```
USE Sue INDEX Sue NEW
TOTAL ON &(INDEXKEY(0)) FIELDS ertek TO Osszeg
```

Helye: CLIPPER.LIB

Lásd még: AVERAGE, INDEX, SORT, SUM

TYPE

Szövegállomány tartalmának megjelenítése

TYPE <állománynév> [TO PRINTER] [TO FILE <új állomány>]

<állománynév> — a megjelenítésre kerülő állomány neve, kiterjesztéssel együtt. Tartalmazhat meghajtót és elérési útvonalat is. Megadhatjuk közvetlenül, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. A kiterjesztést kötelező megadni!

TO PRINTER — a megjelenítés a nyomtatóra is.

TO FILE <új állomány> — a megjelenítés a megadott állományba is történik. Tartalmazhat meghajtót és elérési útvonalat is. Megadhatjuk közvetlenül, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. Ha kiterjesztést nem adunk meg, az alapértelmezés: .txt.

A TYPE parancs segítségével a megadott szövegállományt képernyőn, szükség esetén nyomtatón és/vagy lemezállományban is megjeleníthetjük. Nyomtatás vagy lemezre írás esetén a képernyőn történő megjelenítés a SET CONSOLE OFF paranccsal tiltható le.

Ha az <állománynév> nem tartalmaz meghajtó és/vagy elérési útvonal definíciót, az állomány keresése először az aktuális (DEFAULT) meghajtón és könyvtárban, majd az érvényes SET PATH szerinti elérési útvonalakon történik. Ha az <új állomány> nem tartalmaz meghajtó és/vagy elérési útvonal definíciót, akkor az aktuális (DEFAULT) könyvtárba kerül.

A TYPE paranccsal semmilyen különleges kiegészítést nem adhatunk az állomány tartalmához: nem lehet fejléceket és lapszámozást alkalmazni.

A megjelenítés ideiglenes leállítása a Ctrl-S lenyomásával lehetséges, az Esc billentyű hatástalan.

Példa:

A TYPE parancs alkalmazása:

```
TYPE Sue.prg TO PRINTER
```

Helye: EXTEND.LIB

Lásd még: COPY FILE, SET DEFAULT, SET PATH, SET PRINTER

UNLOCK

Állomány-, illetve rekordzárolás feloldása

UNLOCK [ALL]

ALL — a felhasználó által lezárt összes állomány és rekord felszabadítása. Ha nincs megadva, akkor csak az aktuális munkaterületen zárolt állományok és/vagy rekordok zárolása szünik meg.

Az UNLOCK hálózati parancs, amellyel a felhasználó saját zárolásait oldhatja fel. Használata akkor célszerű, amikor nem akarunk új zárolási (lock) parancsot kiadni. Egyébként egy új FLOCK(), illetve RLOCK() először felszabadítja a korábban zárolt állományt, és csak ezután zárolja az újat.

UNLOCK után a közös használatúvá vált adatbázis, illetve az indexállomány(ok) és a memo aktuális állapota a DOS és más felhasználók számára csak egy COMMIT parancs, vagy az állomány lezárása után biztosított!

A SET RELATION paranccsal összekötött adatbázisokat egy UNLOCK paranccsal automatikusan nem lehet feloldani, csak akkor, ha az UNLOCK ALL formát használjuk.

Példák:

A példában egy falhasználói függvény végrehajtását a rekordzárolás sikerességétől tesszük függővé:

```
USE Sue INDEX Sue SHARED NEW
IF RLOCK()
   Akarmi()
   UNLOCK
ELSE
   ? "Sikertelen függvényhívás"
   BREAK
ENDIF
```

A következő példában a PACK utasításhoz kötelező a teljes állomány zárolása. A PACK végrehajtása után az állomány zárolását feloldjuk:

```
USE Sue INDEX Sue SHARED NEW
IF FLOCK()
   PACK
   UNLOCK
ELSE
   ? "Sikertelen PACK"
   BREAK
ENDIF
```

Helye: CLIPPER.LIB

Lásd még: SET RELATION, USE, FLOCK(), RLOCK()

UPDATE

Az aktuális adatbázis módosítása másik adatbázis alapján

UPDATE FROM <aliasnév> ON <kulcs> [RANDOM]
REPLACE <mező1> WITH <kifejezés1> [,<mező2> WITH <kifejezés2>...]

FROM <aliasnév> — annak a munkaterületnek az azonosítója, amelyen a módosítás alapját képező adatbázis található. Megadhatjuk közvetlenül, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

ON <kulcs> — a módosító adatbázisól a <kulcs>-nak megfelelő kulcsértékű rekordok vesznek rászt a módosításban.

REPLACE <mezőx> — az aktuális munkaterület azon mezői, amelyek a módosítás során új értéket kapnak.

WITH <kifejezésx> — az aktuális mező számára átadásra kerülő érték kifejezése. Ha a módosító adatbázis valamely mezőjére hivatkozunk, azt a munkaterület azonosítójával együtt kell jelölni (lásd példa).

RANDOM — megadása esetén a módosító adatbázis tetszés szerinti sorrendű lehet. Ebben az esetben azonban az aktuális adatbázisnak kötelezően a <kulcs> szerinti rendezettségben kell lennie.

Az UPDATE az aktuális munkaterületen megnyitott adatbázis megadott mezőinek értékét módosítja egy másik munkaterületen nyitott adatbázis mezőinek felhasználásával. Ezzel a paranccsal csak az aktuális munkaterület mezőinek módosítása hajtható végre. Az aktuális és a módosító adatbázis rekordjai között egy rekord-egy rekord", vagy egy rekord-több rekord" relációs kapcsolat állhat fenn. Ez azt jelenti, hogy az aktuális adatbázisban csak az egyedi kulcsértékkel rendelkező rekordok módosítása megy végbe; ha több rekord rendelkezik ugyanazzal a kulccsal, akkor ezek közül az első módosítása történik meg. A módosító adatbázisban tetszőlegesen fordulhat elő azonos kulcsú rekord is.

Ha a RANDOM paraméter nincs megadva mind az aktuális, mind a módosító adatbázisnak kötelezően a <kulcs> szerinti rendezettségben (INDEX vagy SORT után) kell állni. Ha a RANDOM paramétert megadtuk, akkor csak az aktuális adatbázist kell a <kulcs> szerint indexelni, a módosító adatbázis tetszés szerinti sorrendben állhat.

Hálózatban az aktuális adatbázis zárolása az FLOCK() függvénnyel, vagy a USE ... EXCLUSIVE megnyitással kötelező! A módosító adatbázis tetszőlegesen nyitható meg.

Fontos! Ha a DELETED ON állapotban van, akkor az aktuális adatbázis minden rekordja módosításra kerül, beleértve a logikailag törölteket is. A módosító adatbázis logikailag törölt rekordjai ugyanakkor nem vesznek részt a módosításban.

Példa:

A Sue adatbázis *ertek* mezőjét módosítjuk a Dedi adatbázis *ar* mezőértékének felhasználásával:

```
USE Dedi NEW
USE Sue INDEX Sue NEW
UPDATE FROM Dedi ON kulcs;
   REPLACE ertek WITH ertek + Dedi->ar RANDOM
```

Helye: CLIPPER.LIB

Lásd még: INDEX, JOIN, REPLACE, SET UNIQUE*, SORT, TOTAL

USE

Egy adatbázis és kapcsolódó állományai megnyitása

USE [<adatbázisnév> [INDEX <indexállomány-lista>]
[ALIAS <aliasnév>] [EXCLUSIVE | SHARED] [NEW] [READONLY]]

<adatbázisnév> — a megnyitni kívánt adatbázis neve. Megadhatjuk közvetlenül, vagy zárójelbe tett karakteres kifejezéssel helyettesíthetjük. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk.

INDEX <indexállomány-lista> — az aktuális munkaterületen az adatbázishoz megnyitni kívánt legfeljebb 15 indexállomány megnevezése. Az indexállományok neveit megadhatjuk közvetlenül, vagy azt zárójelbe tett karakteres kifejezéssel helyettesíthetjük. Ez utóbbi használatával gyorsabb végrehajtáshoz és rövidebb programhoz juthatunk. Az elsőként megadott indexállomány lesz automatikusan az ellenőrző index. Ha az <indexállomány-lista>-t változó értékeként adjuk meg, és annak értéke NIL, vagy csupa szóköz, az figyelmen kivül marad.

ALIAS <aliasnév> — a munkaterülethez definiált azonosító név. Ha nem adjuk meg, akkor az azonosító az adatbázis neve lesz.

EXCLUSIVE — megadása esetén hálózatban az adatbázis kizárólagos használatra lesz megnyitva. A többi felhasználó számára az adatbázishoz fordulás mindaddig eredménytelen elsz, amíg az állomány lezárásra nem kerül.

SHARED — az adatbázis a hálózatban közös használatra lesz megnyitva. Ha mind az EXCLUSIVE, mind a SHARED paramétert megadjuk, ez utóbbi lesz érvényben.

NEW — az adatbázis a soron következő szabad munkaterületen lesz megnyitva, egyidejűleg ez válik az aktuális munkaterületté. Ha ezt a paramétert nem adjuk meg, akkor az adatbázis az éppen aktuális munkaterületen lesz megnyitva.

READONLY — az adatbázist csak olvasásra nyitja meg. Ez teszi lehetővé a "read-only" attribútummal ellátott állományok megnyitását. Ha az állomány ezzel a paraméterrel nem nyitható meg, az hibajelzéshez vezet. Ha a READONLY paramétert nem adjuk meg, akkor az adatbázis irás-olvasásra lesz megnyitva.

Ha a USE parancsot önmagában, paraméterek nélkül adjuk meg, akkor az aktuális mukaterületen nyitott állomány lezárásra kerül.

A USE parancs az aktuális vagy a következő szabad munkaterületen megnyitja a megadott adatbázist (.dbf), annak memo (.dbt) állományát, valamint a megadott indexállomány(oka)t (.ntx, .ndx). A Clipperben — a DOS 3.3 és annál magasabb verziókban — 250 munkaterületen maximum 255 különböző állomány lehet egyidejűleg nyitva. Mielőtt a USE megnyitja a megadott adatbázist, és az ahhoz kapcsolódó állományokat, előtte lezárja a munkaterületen eddig nyitott állományokat. Az adatbázis első megnyitásakor a rekordmutató az első (nyitott indexállomány esetén az indexkulcs szerint első) rekordra mutat.

Hálózatban az adatbázis vagy az EXCLUSIVE, vagy a SHARED paraméterrel nyitható meg. EXCLUSIVE esetén az állomány lezárásáig más felhasználó az adatbázis rekordjaihoz nem férhet hozzá, míg SHARED esetén azt közösen használhatják. A Clipperben a menetközbeni zárolást az FLOCK() (teljes állomány zárolása), illetve az RLOCK() (rekord zárolása) függvényekkel hajthatjuk végre. E függvények hatása az állomány lezárása nélkül megszüntethető az UNLOCK paranccsal. Ha sem az EXCLUSIVE, sem a SHARED paraméter nincs megadva, akkor a megnyitás az adatbázishoz definiált EXCLUSIVE vagy SHARED módon történik. A Clipper 5.0 verziótól kezdve minden adatbázishoz előre definiálható az automatikus megnyitási mód. A használati mód a SET EXCLUSIVE parancs szerinti állítással is beállítható, ez azonban kevésbé ajánlott módszer.

Hálózatban az indexelt állományok két lépésben történő megnyitását javasoljuk: Az első lépésben magát az állományt nyitjuk meg a USE <adatbázisnév> paranccsal, majd a NETERR() függvény értékének megvizsgálásával megállapíthatjuk, hogy az állományt más felhasználó nem veszi-e kizárólagos használatra igénybe. Ha a NETERR() függvény értéke *hamis* (.F.), akkor az adatbázis számunkra is hozzáférhető, és megnyithatjuk a SET INDEX TO parancs segítségével a kívánt indexállomány(oka)t is.

Az indexlistában felsorolt indexállományok közül megnyitáskor automatikusan az első helyen szereplő lesz az ellenőrző index. Ennek cseréjét — az állományok lezárását és újbóli megnyitását megtakarítva — a SET ORDER paranccsal hajthatjuk végre.

Az aktuális munkaterületen megnyitott adatbázis, és kapcsolódó állományai lezárását a paraméterek nélküli USE, illetve CLOSE paranccsal végezhetjük el. Ha az összes munkaterületen egyidejűleg kívánjuk az adatbázisokat lezárni, ahhoz a CLOSE DATABASES parancsot alkalmazzuk! Ha az aktuális munkaterületen csak az indexállományokat kívánjuk lezárni, az adatbázis nyitott állapotának fenntartása mellett, akkor a CLOSE INDEX, vagy a SET INDEX TO parancsokat adjuk ki.

Fontos tudnivalók:

A Clipper programban egyidejűleg nyitva tartható állományok számát a CONFIG.SYS állomány FILES parancsa, és a Clipper környezeti változójának F paramétere is szabályozza. Az F paraméter azt szabályozza, hogy az aktuális Clipper programban egyidejűleg hány állomány lehet megnyitva. A két paraméter eltérése esetén a Clipper program számára az F paraméter értéke a mérvadó. Ha például a FILES értéke 120, az F paraméter értéke pedig 50, akkor csak 50 állományt nyithatunk meg a Clipperben, hiába engedélyezne a DOS 120-at!

Az egyidejűleg nyitvatartható állományok számát az operációs rendszer is meghatérozza. A DOS 3.3 alatti verzióiban maximum 20 állományt lehetett egyidejúleg megnyitni. ez a szám a 3.3 verzió fölött 255.

Ugyanazt az adatbázist két különböző munkaterületen elvileg megnyithatjuk (különböző ALIAS nevek megadásával), de alkalmazását nem javasoljuk!

Példák:

Az ismertetett többlépcsős hálózati állománymegnyitási módszer:

```
USE Dedi SHARED NEW
IF !NETERR()
   SET INDEX TO Dedi, Lise
ELSE
   ? "Sikertelen megnyitás"
   BREAK
ENDIF
```

A megnyitás zárójelbe tett változókkal (mind az adatbázis, mind az indexállományok neveit változók tartalmazzák):

```
adatb = "Dedi"
indx = {"Dedi", "Lise", "Sue"}
USE (adatb) INDEX (indx[1]), (indx[2]), (indx[3])
```

Helye: CLIPPER.LIB

Lásd még: CLOSE, SELECT, SET INDEX, NETERR(), USED()

WAIT*

A program leállítása egy billentyű lenyomásáig

WAIT [<szöveg>] [TO<változó>]

<szöveg> — tetszőleges típusú kifejezés, amely megjelenik a képernyőn. Ha nem adjuk meg, akkor a "*Press any key to continue...*" ("Nyomjon meg egy billentyűt a folytatáshoz...") szöveg jelenik meg.

TO <változó> — annak a változónak a neve, amely karakteres formában megkapja a lenyomott billentyű értékét. Ha a <változó> még nincs definiálva, vagy a program adott helyén nem látható", akkor *private* változóként létrejön, és megkapja a megadott értéket.

A WAIT parancs hatására a program egy soremelés után kiírja a <szöveg>-et és várakozó állapotba kerül mindaddig, amíg a felhasználó egy billentyűt le nem nyom. Ha a TO paramétert megadtuk, akkor a lenyomott billentyű karakteres értékét a <változó> kapja meg. Ha egy Alt vagy Ctrl kódot adunk meg, akkor a WAIT a <változó>-nak a CHR(0) értéket adja át. Ha az Alt-billentyűzeten egy nem-alfanumerikus értéket adunk meg, akkor az így definiált karakter értéke kerül a változóba. Ha a beadott karakter megjeleníthető, az láthatóvá válik a képernyőn is. A funkcióbillentyűk lenyomása figyelmen kívül marad, hacsak nem definiáltuk egy SET FUNCTION vagy egy SET KEY paranccsal.

A WAIT a korábbi verziókkal való kompatibilitás érdekében maradt meg, alkalmazása nem javasolt! Helyette alkalmazzuk inkább a @...GET/READ parancsokat, vagy egyetlen karakter bekérésére az INKEY() függvényt.

Fontos! A képernyőn megjelenő szöveg — az előzetes soremelés miatt — elrontja" (felfelé görgeti) a képernyő tartalmát. Ezért ha nem kívánunk szöveget megjeleníteni, akkor használjuk a WAIT "" vagy az INKEY(0) alakot, vagy a SET CONSOLE OFF paranccsal tíltsuk le a képernyős megjelenítést.

Példa:

```
WAIT "Nyomj le valamit..." TO valasz
```

Helye: CLIPPER.LIB

Lásd még: @...GET, ACCEPT, INPUT, MENU TO, INKEY()

ZAP

Az aktuális adatbázis összes rekordjának fizikai törlése

ZAP

A ZAP parancs fizikailag törli az aktuális munkaterületen megnyitott adatbázis tartalmát, beleértve a nyitott index- és memo állományokat is. Az álloámny által eddig elfoglalt lemezterület az operációs rendszer számára felszabadul. A ZAP parancs hatása megegyezik a DELETE ALL utáni PACK paranccsal.

Hálózatban a ZAP kiadásához az állományt kizárólagos használatra kell megnyitni (USE ... EXCLUSIVE).

Példa:

```
USE Sue EXCLUSIVE NEW
IF !NETERR()
   SET INDEX TO Sue, Dedi, Kate
   ZAP
   CLOSE Sue
ELSE
   ? "Sikertelen törlés"
   BREAK
ENDIF
```

Helye: CLIPPER.LIB

Lásd még: DELETE, PACK, USE

Clipper függvények

A függvények ismertetése rövidített formában történik. Ennek elsősorban terjedelmi okai vannak, remélhetőleg azonban a paraméterek, és a visszaadott értékek magyarázata elég támpontot ad alkalmazásukhoz. A legtöbb függvény már ismert a korábbi verziókból, azonban a gyakorlott programozóknak sem árt megnézni a lehetőségeket, mert több függvény megváltozott (általában bővebb) tartalommal rendelkezik. Sajnos helyhiány miatt el kellett tekinteni a példák közlésétől is, ami sokszor többet mond minden magyarázatnál; ezeket az eredeti dokumentációban, vagy a Norton Guide szerinti on-line dokumentációban tekinthetik meg. A függvények előfordulási helyét az előszerkesztési könyvtár összeállításához nyújtandó segítségként adjuk meg, amelyeknél nincs feltüntetve, azt a CLIPPER.LIB tartalmazza. Az ismertetésben szerepelnek azok a függvények is, amelyek az eredeti dokumentációból kimaradtak.

AADD()

Egy új elem toldása a tömb végéhez

AADD(<tömb>, <értékkifejezés>) --> érték

<tömb> — annak a tömbnek a neve, amelyet az új elemmel kiegészítünk.

<értékkifejezés> — az új elem értéke, amely lehet egy altömb is.

Visszaadott érték:

- az <értékkifejezés> értéke;
- NIL, ha az <értékkifejezés> nincs megadva.

Lásd még: AINS(), ASIZE()

ABS()

Numerikus kifejezés abszolút értékének meghatározása

ABS(<numerikus kifejezés>) --> pozitív szám

<numerikus kifejezés> — az a numerikus kifejezés, amelynek az abszolút értékét kívánjuk meghatározni.

Visszaadott érték: Egy pozitív szám, vagy nulla.

ACHOICE()

Feltáruló menü létrehozása

ACHOICE(<felső>, <bal>, <alsó>, <jobb>, <tömbnév1>
[<tömbnév2> | <logikai kifejezás>], [<saját függvény>],
[<tömbindex>], [<ablaksor>]) --> indexsorszám

<felső>, <bal>, <alsó>, <jobb> — meghatározza a képernyőn azt az ablakot, amelyben a megjelenítés történik.

<tömbnév1> — egy karakteres kifejezéseket tartalmazó tömb neve, ahol az egyes tömbelemek menüpontokként jelennek meg.

<tömbnév2> | <logikai kifejezás>- egy mutató jellegű tömb, amelynek minden eleme egy logikai érték, vagy egy karaktersorozat. (Ez utóbbi esetben a tömbelemek olyan makróértékeket tartalmaznak, amelyek kiértékelése után logikai adattípust kapunk.) Az egyes tömbelemek azt mutatják meg, hogy a <tömbnév1> megfelelő tömbelemei kiválaszthatók-e vagy sem. Ha összesen csak egy paramétert adunk meg, akkor minden elemre ennek az értéke vonatkozik. Alapértelmezésként minden menüpont kiválasztható.

<saját függvény> — az egyes lenyomott billentyűkhöz egy-egy saját függvény illeszthető. A függvény nevét zárójelek és argumentumok nélkül kell megadni.

<tömbindex> — a menü alapértelmezés szerinti kiválasztott eleme. (Az a menüpont, amely a menü megjelenésekor kijelölt (kivilágított) állapotban lesz.) Ha nem adjuk meg, akkor az első tömbelem.

<ablaksor> — az ablaknak az a sora, amelyben az alapértelmezés szerint kiválasztott elem (<tömbindex>) megjelenik. A sorok számozása az ablakon belül nullától kezdődik. Alapértelmezés az ablak legfelső sora (0.sor). Ennek a paraméternek a segítségével több menüszerkezet is megjeleníthető egy ablakban.

Visszaadott érték: Az ACHOICE() a <tömbnév1> tömb azon index sorszámát adja vissza, amelyik kiválasztásra került. Ha a kiválasztási folyamat megszakadt, akkor a visszaadott érték nulla. (Például Esc nyomása esetén.)

Az ACHOICE() navigáló billentyűi

Billentyű	Hatás
**felfelé nyíl	az előző menüpontra
**lefelé nyíl	a következő menüpontra
Home	az első menüpontra
End	az utolsó menüpontra
**Ctrl-Home	az ablakban látható első menüpontra
**Ctrl-End	az ablakban látható utolsó menüpontra
**PgUp	lapozás egy ablaknyival vissza
**PgDn	lapozás egy ablaknyival előre
**Ctrl-PgUp	az első menüpontra
**Ctrl-PgDn	az utolsó menőpontra
Return	a menüpont kiválasztása, befejezés
Esc	befejezés kiválasztás nélkül
balra nyíl	befejezés kiválasztás nélkül
jobbra nyíl	befejezés kiválasztás nélkül
kezdőbetű	az ilyen betűvel kezdődő következő menüpontra

A **jelölt billentyűk használata saját függvény alkalmazása esetén is megengedett.

Saját függvény készíthető, ha hierarchikus menürendszert akarunk létrehozni, vagy át kívánjuk definiálni a billentyűket. A fenti táblázatban **-gal jelölt billentyűk kivételével a felhasználói függvény minden egyéb billentyű lenyomását átdefiniálhatja saját igényei szerint. Felhasználói függvény definiálása esetén az ACHOICE() három paramétert ad át:

- Az aktuális ACHOICE() módja
- Az aktuális tömbelem sorszáma
- A menüablak azon sorának értéke, ahol a kiválasztás történt

Az ACHOICE() módjának értékei

Mód	Achoice.ch	Tartalma
0	AC_IDLE	Semmi nem történt (kurzormozgató billentyű nyomása)
1	AC_HITTOP	A tömb elején felfelé mozgást kisérelt meg
2	AC_HITBOTTOM	A tömb alján lefelé mozgást kisérelt meg
3	AC_EXCEPT	Billentyű lenyomása történt
4	AC_NOITEM	Nem kiválasztható menüpont

A felhasználói függvény által visszaadható értékek

Érték	Achoice.ch	Tartalma
0	AC_ABORT	A kiválasztás vége (az ACHOICE() befejezi működését és 0 értéket ad vissza)
1	AC_SELECT	A kiválasztás végrehajtása (az ACHOICE() a kiválasztott tömbelem indexének sorszámát adja vissza)
2	AC_CONT	Az ACHOICE() folytatása
3	AC_GOTO	A következő olyan menüpontra áll, amelynek első betűje megegyezik a lenyomott billentyűvel (ha ilyen nincs, nem csinál semmit)

Helye: EXTEND.LIB, kiegészítő állományok: Achoice.ch és Inkey.ch

Lásd még: MENU TO, SET COLOR*

ACLONE()

Egymásbaágyazott vagy többdimenziós tömbök duplikálása

ACLONE(<forrástömb>) --> másolt tömb

<forrástömb> — a duplikálni kívánt tömb neve.

Visszaadott érték: a <forrástömb> másolata.

Lásd még: ACOPY(), ADEL(), AINS(), ASIZE()

ACOPY()

Elemek másolása egyik tömbből egy másikba

ACOPY(<forrástömb>, <céltömb>, [<kezdőindex>], [<elemszám>], [<cél-kezdőindex>]) --> céltömb

<forrástömb> — annak a tömbnek a neve, ahonnan a másolás történik.

<céltömb> — annak a tömbnek a neve, ahová a másolás történik.

<kezdőindex> — annak az indexnek a sorszáma a forrástömbben, amelytől a másolás történik. Alapértelmezése: 1.

<elemszám> — azt adja meg, hogy a <kezdőindex>-től kezdve hány elem másolása történjen meg. Ha nincs megadva, akkor a <kezdőindex>-től (vagy ennek hiányában a tömb elejétől) az összes tömbelem átmásolásra kerül.

<cél-kezdőindex> — a céltömbben annak az elemnek a sorszáma, amelytől kezdve a másolás történik. Alapértelmezés: 1.

Visszaadott érték: A <céltömb> új értéke.

Lásd még: ACLONE(), ADEL(), AEVAL(), AFILL(), AINS(), ASORT()

ADEL()

Tömbelem törlése

ADEL(<tömbnév>, <indexsorszám>) --> új tömbtartalom

<tömbnév> — annak a tömbnek a neve, amelynek valamely elemét törölni kívánjuk.

<indexsorszám> — a törölni kívánt tömbelem indexe.

Visszaadott érték: A tömb törlés utáni állapotban, az <indexsorszám> utáni elemek balra tolódnak, az utolsó értéke NIL lesz.

Lásd még: ACOPY(), AFILL(), AINS()

ADIR()*

Tömb feltöltése könyvtárinformációkkal

**ADIR([<állománymaszk>],[<állománynévtömb>],[<mérettömb>],
[<dátumtömb>],[<időtömb>],[<attributumtömb>]) --> állományok száma**

<állománymaszk> — az aktuális (DEFAULT) könyvtárban, vagy a megadott meghajtón, illetve könyvtárban kiválasztásra kerülő állományok meghatározása, amelyhez felhasználhatók a csoportképző (?, *) karakterek is. Az alapértelmezés: *.*.

<állománynévtömb> — az állománymaszk alapján kiválasztott állományok neveit és kiterjesztéseit karakteres formában, csupa nagybetűvel tartalmazó tömb.

<mérettömb> — az állománymaszk alapján kiválasztott állományok méreteit numerikus formában tartalmazó tömb.

<dátumtömb> — az állománymaszk alapján kiválasztott állományok felírásának (utolsó módosításának) dátumait tartalmazó tömb, melynek minden eleme dátum típusú.

<időtömb> — az állománymaszk alapján kiválasztott állományok felírásának (utolsó módosításának) idejeit tartalmazó tömb, amelynek minden eleme karakteres típusú óó:pp:mm alakban.

<attributumtömb> — az állománymaszk alapján kiválasztott állományok attributumait karakteres formában tartalmazó tömb. Ha az <attributumtömb> paramétert megadjuk, akkor a

kiválasztási maszknak megfelelő összes létező állomány megjelenik, beleértve a rejtett és a rendszerállományokat, valamint a könyvtárneveket. Ha nincs megadva, akkor csak a normál állományok jelennek meg.

Az attributumértékek:

R csak olvasható (read-only);
H rejtett (hidden);
S rendszer (system);
D könyvtár (directory);
A archív.

Visszaadott érték: A maszknak megfelelő állományok számát adja vissza numerikus értékként.

Helye: EXTEND.LIB

Lásd még: ACHOICE(), AEVAL(), ASCAN(), ASORT(), DIRECTORY(), LEN()

AEVAL()

Kódblokk kialakítása tömbelemekből

AEVAL(<tömb>, <kódblokk>, [<kezdőindex>], [<elemszám>]) --> tömb

<tömb> — a tömb neve.

<kódblokk> — az elemek információi számára létrehozott kódblokk.

<kezdőindex> — a kezdő elem (alapértelmezés 1).

<elemszám> — a <kezdőindex>-től számított elemek száma. Ha nincs megadva, akkor az összes elem.

Visszaadott érték: a <tömb>-re vonatkozó információk.

Az AEVAL() függvény kiértékel egy kódblokkot, amelynek elemei — mint blokkparaméterek — egy tömböt alkotnak. A műveletben a tömb minden eleme résztvesz, hacsak a <kezdőindex> és az <elemszám> paraméterekkel mást nem definiálunk. Az AEVAL() a tömb tartalmát nem módosítja, csak átadja a kódblokknak. Az AEVAL() ugyanazt a funkciót látja el, mint a DBEVAL() adatbázis-rekordok esetében.

Lásd még: DBEVAL(), EVAL(), QOUT()

AFIELDS()*

Tömb feltöltése az aktuális adatbázis szerkezetével

AFIELDS([<mezőnévtömb>], [<típustömb>], [<hossztömb>],
[<tizedesjegytömb>]) --> mezők száma

<mezőnévtömb> — a mezőnevekkel feltöltésre kerülő tömb neve. Minden eleme karaktersorozat.

<típustömb> — az egyes mezők típusaival feltöltött tömb neve. Minden eleme karaktersorozat.

<hossztömb> — az egyes mezők hosszaival feltöltött tömb neve. Minden eleme numerikus.

<tizedesjegytömb> — az egyes mezők tizedesjegyeinek számával feltöltött tömb neve. Minden eleme numerikus. Ha a mező típusa nemnumerikus, akkor az illető elem értéke nulla.

Visszaadott érték: a mezők száma, vagy a megadható tömbök közül a legkevesebb elemmel rendelkező elemeinek a száma, ha az kisebb a mezők számánál.

Helye: EXTEND.LIB

Lásd még: ACHOICE(), ADIR(), AEVAL(), ASCAN(), ASORT(), DBSTRUCT(), FCOUNT(), FIELD(), LEN(), TYPE()

AFILL()

A tömb elemeinek feltöltése egy megadott értékkel

AFILL(<tömbnév>, <érték>, [<kezdőindex>], [<elemszám>]) --> tömb

<tömbnév> — az értékkel feltöltendő tömb neve.

<érték> — a feltöltő érték, amely tetszőleges típusú kifejezés lehet.

<kezdőindex> — az a tömbindex, amelytől a feltöltést kezdeni kívánjuk. Alapértelmezés: 1.

<elemszám> — a <kezdőindex>-től számítva ennyi elem kerül feltöltésre. Ha nincs megadva, akkor a tömb végéig érvényes.

Visszaadott érték: a tömb új értékkel feltöltve.

Lásd még: AADD(), AEVAL(), DBSTRUCT(), DIRECTORY()

AINS()

Egy NIL elem beszúrása a tömb tetszőleges helyére

AINS(<tömbnév>,<pozíció>) --> tömb

<tömbnév> — az eredeti tömb, amelybe egy új elemet kell beszúrni.

<pozíció> — az a hely, ahová az új elem beszúrásra kerül.

Visszaadott érték: az új elemmel kiegészített, de az eredetivel azonos hosszúságú tömb. A <pozíció>-tól minden elem jobbra tolódik, az utolsó elvész.

Lásd még: AADD(), ACOPY(), ADEL(), AEVAL(), AFILL(), ASIZE()

ALIAS()

A munkaterület nevének lekérdezése

ALIAS([<munkaterület száma>]) --> aliasnév

<munkaterület száma> — tetszőleges munkaterületszám.

Visszaadott érték: Karaktersorozatként adja vissza a kívánt számú munkaterület aliasnevét. Ha a megadott számú munkaterület nem létezik, akkor az aktuális munkaterület nevével tér vissza. Ha a megadott munkaterületen nincs megnyitott adatbázis, akkor üres karaktersorozatot ("") kapunk.

Lásd még: SELECT(), SELECT, USE

ALLTRIM()

Karaktersorozat bevezető és záró szóközeinek levágása

ALLTRIM(<karakteres kifejezés>) --> tömör karaktersorozat

<karakteres kifejezés> — a tömöríteni kívánt karakteres kifejezés.

Visszaadott érték: karakteres kifejezés, amelynek a bevezető és végső szóközeit a függvény levágta.

Helye: EXTEND.LIB

Lásd még: LTRIM(), PADx(), RTRIM()

ALTD()

A Clipper Debugger indítása

ALTD([<mód>]) --> NIL

<mód> — a Debugger működési módjának beállítása. Értékei:

Argumentum	Működés
nincs megadva	a debugger indítása
0	nem engedi meg az Alt-D használatát
1	megengedi, hogy az Alt-D-vel indítsuk a Debuggert
egyéb	definiálatlan

Visszaadott érték: mindig NIL.

Lásd még: SETCANCEL(), SET ESCAPE

ARRAY()

Egy definiálatlan elemekkel rendelkező tömb létrehozása

ARRAY(<elemszám> [, <elemszám>...]) --> tömb

<elemszám> — a dimenziónként definiált elemek száma. Maximális értéke dimenziónként 4096 lehet; a dimenziók számára korlátozás nincs (azt csak a memória mérete korlátozza).

Visszaadott érték: a megadott dimenzióméretű tömb definiálatlan elemekkel.

Lásd még: AADD(), ACLONE(), ACOPY(), ADEL(), AEVAL(), AFILL(), AINS(), ASCAN(), ASIZE(), ASORT(), LOCAL, PRIVATE, PUBLIC, STATIC

ASC()

Karakter ASCII kódjának meghatározása

ADC(<karakters kifejezés>) --> ASCII-kód

<karakteres kifejezés> — a megadott karakteres kifejezés, amelynek első karakterét veszi a függvény fiogyelembe.

Visszaadott érték: a <karakteres kifejezés> első karakterének ASCII kódja, amely 0 és 255 közötti szám lehet.

Lásd még: CHR(), INKEY(), STR(), VAL()

ASCAN()

Tömbön belüli keresés

ASCAN(<tömbnév>, <keresett érték>,
[<kezdőindex>], [<elemszám>]) --> talált index

<tömbnév> — a tömb neve, amelyben egy elemet keresünk.

<keresett érték> — az az érték, amelyet a tömbön belül keresünk. Lehet egyszerű érték, vagy kódblokk. Az egyszerű érték lehet karakteres, dátum, logikai vagy numerikus típusú.

<kezdőindex> — az az elem, amelytől a keresést kezdjük. Alapértelmezése: 1.

<elemszám> — a <kezdőindex>- től számított ennyi elemen keresztül folyik a keresés. Ha nincs megadva, a tömb végéig érvényes.

Visszaadott érték: a keresett értékkel megegyező első elem indexe, ha nincs ilyen elem, akkor 0. Ha a keresett érték kódblokk, akkor annak az elemnek az indexét adja, amelynél a blokk igaz (.T.) értéket ad vissza.

Lásd még: AEVAL(), EVAL()

ASIZE()

Tömbméret módosítása

ASIZE(<tömb>, <új méret>) --> új tömb

<tömb> — a növelni vagy csökkenteni kívánt tömb neve.

<új méret> — a tömb új mérete.

Visszaadott érték: az új mérettel rendelkező tömb.

Lásd még: AADD(), ADEL(), AFILL(), AINS()

ASORT()

Tömbelemek rendezése

ASORT(<tömbnév>, [<kezdőindex>], [<elemszám>], [<szabály>]) --> tömb

<tömbnév> — a rendezni kívánt tömb neve.

<kezdőindex> — annak az elemnek az indexe, ahonnan a rendezés kezdődik. Ha nem adunk meg semmit, értéke 1.

<elemszám> — a <kezdőindex>-től számítva ennyi elem vesz részt a rendezésben. Ha nincs megadva, a tömb végéig érvényes.

<szabály> — a rendezés sorrendjét meghatározó kódblokk. Ha nincs megadva, az alapértelmezés a növekvő sorrend.

Visszaadott érték: a tömb rendezett alakja.

Helye: EXTEND.LIB

Lásd még: ASCAN(), EVAL(), SORT

AT()

Egy karaktersorozat helyének keresése egy másikon belül

AT(<mit>, <miben>) --> kezdőpozíció

<mit> — az a karaktersorozat, amelynek előfordulási helyét keressük.

<miben> — az a karaktersorozat, amelyben a <mit> keressük.

Visszaadott érték: egész számú numerikus érték, amely azt mutatja, hogy a keresett karaktersorozat előfordulási helye a <miben> hányadik pozícióján kezdődik. Ha nem fordul elő benne, akkor értéke 0.

Lásd még: RAT(), STRTRAN(), SUBSTR()

BIN2I()

16 bites egész szám konvertálása numerikusra

BIN2I(<karakteres kifejezés>) --> szám

<karakteres kifejezés> — egy binárisan tárolt karakteres kifejezés, aminek az első két byte-ját numerikus egész értékre kívánjuk konvertálni.

Visszaadott érték: numerikus egész érték.

Helye: EXTEND.LIB, forrása EXAMPLEA.ASM

Lásd még: BIN2L(), BIN2W(), FREAD(), FREADSTR(), I2BIN(), L2BIN()

BIN2L()

32 bites egész szám konvertálása numerikusra

BIN2L(<karakteres kifejezés> --> szám

<karakteres kifejezés> — egy binárisan tárolt karakteres kifejezés, aminek az első négy byte-ját numerikus egész értékre kívánjuk konvertálni.

Visszaadott érték: numerikus egész érték.

Helye: EXTEND.LIB, forrása EXAMPLEA.ASM

Lásd még: BIN2I(), BIN2W(), FREAD(), FREADSTR(), I2BIN(), L2BIN()

BIN2W()

16 bites egész konvertálása

BIN2W(<karakteres kifejezés>) --> szám

<karakteres kifejezés> — egy 16-bites jelöletlen formában tárolt karakteres kifejezés, aminek az első két byte-ját numerikus egész értékre kívánjuk konvertálni.

Visszaadott érték: numerikus egész érték.

Helye: EXTEND.LIB, forrása EXAMPLEA.ASM

Lásd még: BIN2I(), BIN2L(), FREAD(), FREADSTR(), I2BIN(), L2BIN()

BOF()

Állomány-eleje jel vizsgálata

BOF() --> logikai érték

Visszaadott érték: A BOF() igaz (.T.) értéket ad vissza, ha a rekordmutató az első rekord elé mutat. Ez két esetben lehetséges: vagy az első rekordon állva egy SKIP -1 parancsot adunk ki, vagy az állomány üres (ekkor az EOF() értéke is igaz (.T.) lesz). Ha a munkaterületen nincs nyitott adatbázis, akkor értéke hamis (.F.).

Lásd még: EOF(), SKIP

BREAK()

A BREAK utasítás helyettesítése egy szekvenciában

BREAK(<kifejezés>) --> NIL

<kifejezés> — a RECOVER utasításnak átadandó érték. Tetszőleges adattípus, illetve kódblokk értéke lehet.

Visszaadott érték: mindig NIL.

BROWSE()

Rekordok megjelenítése ablakban

BROWSE([<felső>], [<bal>], [<alsó>], [<jobb>]) -> NIL

<felső>, <bal>, <alsó>, <jobb> — annak az ablaknak a négy sarokpontja, amelyben a megjelenítés történik. Ha nem adjuk meg, akkor az 1, 0, MAXROW(), MAXCOL() értékeket veszi fel.

Visszaadott érték: mindig NIL.

Helye: EXTEND.LIB, forrása EXAMLEP.PRG

Lásd még: DBEDIT()

CDOW()

Dátumból a hét napjának szöveges kiírása (angolul)

CDOW(<dátumkifejezés> --> nap

<dátumkifejezés> — a konvertálni kívánt dátum.

Visszaadott érték: a hétnek az a napja (angolul), amelyre a megadott dátum esik. Az első betű nagy, míg a többi kicsi. Az üres dátumértékre a függvény üres karaktersorozatot ("") ad vissza.

Lásd még: CTOD(), DATE(), DAY(), DOW()

CHR()

Az ASCII kód karakteres formája

CHR(<numerikus kifejezés>) --> karakter

<numerikus kifejezés> — egy ASCII kódérték 0 és 255 között.

Visszaadott érték: egy karakter, amely az ASCII kódnak felel meg.

Lásd még: ASC(), INKEY()

CMONTH()

Dátumból a hónap szöveges kiírása (angolul)

CMONTH(<dátumkifejezés>) --> hónap

<dátumkifejezés> — a konvertálni kívánt dátum.

Visszaadott érték: a megadott dátumban szereplő hónap angol neve, amely nagybetűvel kezdődik, a többi betű kicsi. Az üres dátumkifejezésre üres karaktert ("") ad vissza.

Lásd még: CDOW(), CMONTH(), DATE(), DAY(), MONTH(), YEAR()

COL()

A kurzor oszloppozíciója

COL() --> oszlopszám

Visszaadott érték: numerikus egész érték 0 és MAXCOL() között.

Lásd még: MAXCOL(), PCOL(), PROW(), ROW(), @...CLEAR, @...GET, @...SAY, CLEAR

CTOD()

Karakteres dátumforma dátum típusra konvertálása

CTOD(<karakteres dátum>) --> dátumkifejezés

<karakteres dátum> — egy olyan karakteres kifejezés, amely a dátumnak megfelelően tartalmazza a hónap, nap és év számát, és ezek egymástól egy számtól eltérő jellel vannak elválasztva. Sorrendjüknek a SET DATE formában megadottal kell megegyezni. Ha az évszázadot nem jelöljük, akkor a SET EPOCH szabálya szerint azzal kiegészül.

Visszaadott érték: dátumérték. Ha a <karakteres dátum> hibás dátum, üres karakter (""); ha SPACE(8), akkor üres dátumkifejezést ad vissza (" / / ").

Lásd még: DATE(), DTOC(), DTOS(), SET CENTURY, SAT DATE, SET EPOCH

CURDIR()

Az aktuális DOS könyvtár lekérdezése

CURDIR([<meghajtó-azonosító>]) --> könyvtárnév

<meghajtó-azonosító> — a lemezmeghajtó betűjele. Ha nincs megadva, akkor az aktuális DOS lemezmeghajtó az alapértelmezés.

Visszaadott érték: az aktuális DOS lemezkönyvtár neve karaktersorozatos formában. Az első \ jelet nem írja ki. Hiba esetén, vagy ha az aktuális könyvtár a főkönyvtár, akkor a visszaadott érték egy üres karaktersorozat ("").

Helye: EXTEND.LIB, forrása EXAMPLEA.ASM

Lásd még: FILE()

DATE()

A rendszerdátum lekérdezése

DATE() --> rendszerdátum

Visszaadott érték: dátumformában adja vissza a gép által számolt rendszerdátumot.

Lásd még: CTOD(), DTOC(), DTOS(), SET CENTURY, SET DATE

DAY()

Dátumkifejezésből a nap értéke

DAY(<dátum>) --> nap értéke

<dátum> — a vizsgált dátum.

Visszaadott érték: 0 és 31 közé eső numerikus érték, amely a megadott dátumkifejezésből a napot adja vissza. Ha a dátumkifejezés üres, vagy február 29-e van megadva nem szökőévre, akkor hibajelzés nélkül 0.

Lásd még: CDOW(), CMONTH(), DOW(), MONTH(), STR(), YEAR()

DBCREATE()

Adatbázis létrehozása egy szerkezetét tartalmazó tömbből

DBCREATE(<adatbázisnév>, <szerkezettömb>) --> NIL

<adatbázisnév> — az új adatbázis neve, amelyhez megadható a meghajtó és az elérési útvonal is. Ha kiterjesztést nem adunk meg, az alapértelmezés .dbf.

<szerkezettömb> — az új adatbázis szerkezetét leíró tömb, amely mezőnként egy altömböt alkot. Minden altömb a következőket tartalmazza:

Pozíció	Metajel	Dbstruct.ch
1	cName	DBS_NAME
2	cType	DBS_TYPE
3	nLength	DBS_LEN
4	nDecimals	DBS_DEC

Visszaadott érték: mindig NIL.

A meződefiniálás olvashatóbbá tétele érdekében a Dbstruct.ch tartalmazza a nevek és a tömbindexek közötti megfeleltetést.

Helye: CLIPPER.LIB, kapcsolódó állomány Dbstruct.ch

Lásd még: AFIELDS(), DBSTRUCT(), COPY STRUCTURE EXTENDED, CREATE FROM

DBEDIT()*

Rekordok táblázatos megjelenítése

DBEDIT([<felső>], [<bal>], [<alsó>], [<jobb>],
[<mezőnévtömb>], [<sajátfüggvény-név>],
[<formátumtömb> | <formátumkaraktersor>],
[<fejléctömb> | <fejléckaraktersor>],
[<fejlécaláhúzás-tömb> | <fejlécaláhúzáskarsor>],
[<oszlopelválasztó-tömb> | <oszlopelválasztókar>],
[<lábjegyzetelválasztó-tömb> | <lábjegyzetelválasztó>],
[<lábjegyzettömb> | <lábjegyzetkaraktersor>] --> NIL

<felső>, <bal>, <alsó>, <jobb> — a DBEDIT() ablak sarokpontjainak helye. A sorok száma 0 és MAXROW(), az oszlopok száma 0 és MAXCOL() értéke közé eshet. Ha nem adjuk meg, alapértelmezésként a 0,0, MAXROW(), MAXCOL() értékeket veszi.

<mezőnévtömb> — egy karakteres kifejezéseket tartalmazó tömb, amely az adatbázis megjelenítendő mezőneveit, illetve azokat a kifejezéseket tartalmazzák, amelyeket mint oszlopértékeket minden sorban ki kívánunk írni. Ha nem adjuk meg, az adatbázis összes mezője megjelenik.

<sajátfüggvény-név> — egy felhasználói függvény neve, amelyet akkor kell végrehajtani, ha egy definiálatlan billentyű lenyomása következett be. A függvény nevét karakteres kifejezésként zárójelek és argumentumlista nélkül kell megadni. Erre az argumentumra a DBEDIT() nagyon kényes. Állandó paraméterei a DBEDIT() mód (visszaadott értékeit a táblázat tartalmazza), és az aktuális oszlop indexe.

<formátumtömb> — minden egyes oszlop PICTURE formátumát tartalmazza. A <formátumkaraktersor> megadása esetén minden oszlop azonos formátummal rendelkezik. A PICTURE formátumról bővebben TRANSFORM() függvénynél, illetve a @...SAY parancsnál.

<fejléctömb> — az egyes oszlopok fejlécébe beírni kívánt karaktersorozatokat tartalmazza. Ha a <fejléckaraktersor>-t adjuk meg, akkor minden oszlop azonos fejléccel rendelkezik. Többsoros fejlécet is megadhatunk, ha a törések helyére pontosvesszőt (;) teszünk. Ha nem adjuk meg, akkor a <mezőnévtömb> szerinti értékek kerülnek kiírásra, illetve ha az se lett megadva, akkor az aktuális munkaterület mezői.

<fejlécaláhúzás-tömb> — minden egyes mező oszlopára vonatkozóan definiálhatjuk, hogy mivel legyen a fejléc az adattartalmaktól elválasztva. A <fejlécaláhúzáskarsor> alkalmazása esetén minden oszlop azonosan lesz aláhúzva. Ha nem adunk meg semmit, akkor az aláhúzás duplavonallal történik.

<oszlopelválasztó-tömb> — minden egyes mező oszlopára definiálhatjuk, hogy mivel legyen elválasztva a tőle jobbra álló másik mezőtől. Az <oszlopelválasztókar> alkalmazása esetén ez minden oszlopnál azonos lesz. Ha nem adunk meg semmit, az alapértelmezés a sima vonal.

<lábjegyzetelválasztó-tömb> — minden egyes mező oszlopára vonatkozóan definiálhatjuk, hogy mivel legyen az adattartalmaktól a lábjegyzet elkülönítve. A <lábjegyzetelválasztó> alkalmazása esetén minden oszlop azonosan lesz aláhúzva. Ha nem adunk meg semmit, akkor nincs elkülönítés.

<lábjegyzettömb> — minden oszlophoz külön lábjegyzeteket definiálhatunk. A <lábjegyzet-karsor> alkalmazása esetén minden oszlop azonos lábjegyzettel rendelkezik. Többsoros lábjegyzetek is készíthetők, ha a törés helyére pontosvesszőt (;) teszünk. Ha nem adunk meg semmit, nincs lábjegyzet.

Visszaadott érték: mindig NIL.

A DBEDIT() módok

Állapot	Dbedit.ch	Jelentés
0	DE_IDLE	Egy éppen hatástalan kurzormozgató billentyű lenyomása történt
1	DE_HITTOP	A kurzormozgató billentyű lenyomásával elérte az állomány elejét
2	DE_HITBOTTOM	A kurzormozgató billentyű lenyomásával elérte az állomány végét
3	DE_EMPTY	A munkaterületen nincs rekord
4	DE_EXCEPT	Egy tetszőleges billentyű lenyomása történt (értéke a LASTKEY() függvénnyel megállapítható)

A DBEDIT() sajátfüggvény visszaadott értékei

Érték	Dbedit.ch	Jelentés
0	DE_ABORT	A DBEDIT() megszakítása
1	DE_CONT	A DBEDIT() folytatása
2	DE_REFRESH	A szerkesztés folytatása a képernyő felfrissítésével

Helye: EXTEND.LIB, kapcsolódó állomány Dbedit.ch és Inkey.ch

Lásd még: ACHOICE(), BROWSE(), MEMOEDIT(), TRANSFORM(), @...GET, READ

DBEVAL()

Kódblokk szerinti értékadás a feltételnek megfelelő rekordok számára

DBEVAL(<kódblokk>,[<FOR_blokk>], [<WHILE_blokk>], [<lépésköz>], [<rekordsorszám>], [<folytatás-feltétel>]) --> NIL

<kódblokk> — a feltételnek megfelelő rekordoknál végrehajtandó művelet definiálása.

<FOR_blokk> — a <kódblokk> szerinti műveletek csak a feltételnek megfelelő rekordoknál kerülnek végrehajtásra.

<WHILE_blokk> — a <kódblokk> szerinti műveletek addig kerülnek végrehajtásra, amíg a feltétel igaz.

<lépésköz> — numerikus, annak meghatározására, hogy az aktuálistól kezdődően hány rekord vesz részt a műveletekben (megegyezik a hagyományos NEXT utasítással).

<rekordsorszám> — numerikus, ha megadjuk, akkor a <kódblokk> csak erre a rekordra lesz végrehajtva (megegyezik a hagyományos RECORD utasítással).

<folytatás-feltétel> — logikai, azt határozza meg, hogy a DBEVAL() az összes rekordnál végrehajtásra kerüljön, vagy csak az aktuálistól kezdődően (a normál ALL és REST hatóköröknek megfelelően). Ha értéke igaz (.T.), akkor a REST, egyébként az ALL érvényesül. Alapértelmezése: ALL.

Visszaadott érték: mindig NIL.

Lásd még: AEVAL(), EVAL()

DBF()*

Az aktuális aliasnév (adatbázisnév) lekérdezése

DBF() --> aliasnév

Visszaadott érték: Az aktuális munkaterületen megnyitott adatbázis alias neve. Ha ilyen nincs, akkor üres karaktersorozat ("") kerül visszaadásra.

Helye: EXTEND.LIB, forrásnyelvű listája Examplep.prg

Lásd még: ALIAS(), USE()

DBFILTER()

Az érvényben levő szűrőkifejezés meghatározása

DBFILTER() --> szűrőkifejezés

Visszaadott érték: Az aktuális munkaterületen megnyitott adatbázisra vonatkozóan érvényben levő szűrőkifejezés (filter) karakteres formában.

Lásd még: DBRELATION(), DBRSELECT(), SET FILTER

DBRELATION()

A megadott sorszámú relációs kapcsolókifejezés lekérdezése

DBRELATION(<reláció sorszáma>) --> kapcsolókifejezés

<reláció sorszáma> — az aktuális munkaterületen megnyitott relációs kapcsolatok megfelelő sorszáma (a relációs kapcsolatok számozása a SET RELATION szerinti definiálásuk sorrendjében történik).

Visszaadott érték: egy karakteres kifejezés, amely tartalmazza a megadott sorszámú relációs kapcsolatot. Ha ilyen nincs, akkor üres értékkel ("") tér vissza.

Lásd még: DBFILTER(), DBRSELECT(), SET RELATION

DBRSELECT()

A relációs kapcsolat munkaterületének lekérdezése

DBRSELECT(<reláció sorszáma> --> munkaterület száma

<reláció sorszáma> — az aktuális munkaterületen megnyitott relációs kapcsolatok megfelelő sorszáma (a relációs kapcsolatok számozása a SET RELATION szerinti definiálásuk sorrendjében történik).

Visszaadott érték: numerikus, annak a munkaterületnek a sorszáma, amellyel az aktuális adatbázis relációs kapcsolatban áll. Ha ilyen nincs, a visszaadott érték nulla.

Lásd még: DBFILTER(), DBRELATION(), SET RELATION

DBSTRUCT()

Az adatbázis szerkezetét tartalmazó tömb létrehozása

DBSTRUCT() --> struktúratömb

Visszaadott érték: az aktuális adatbázis szerkezete egy tömbben, amelynek hossza a mezők számával egyezik meg. A tömb minden eleme egy altömb, amely a mezővel kapcsolatos információkat tartalmazza, ennek tartalma:

Pozíció	Metajelölés	Jelentés	Dbstruct.ch
1	cName	mezőnév	DBS_NAME
2	cType	típusa	DBS_TYPE
3	nLength	hossz	DBS_LEN
4	nDecimals	tizedeshossz	DBS_DEC

Helye: CLIPPER.LIB, kapcsolódó állomány: Dbstruct.ch

Lásd még: AFIELDS(), COPY STRUCTURE EXTENDED

DELETED()

Az aktuális rekord töröltségi állapotának lekérdezése

DELETED() --> töröltség jelzése

Visszaadott érték: igaz (.T.), ha az aktuális rekord törlésre ki van jelölve, egyébként hamis (.F.). Ha nincs nyitott adatbázis az aktuális munkaterületen, akkor is hamis (.F.) értéket ad vissza.

Lásd még: DELETE, PACK, RECALL, SET DELETED

DESCEND()

Csökkenő sorrendű indexérték létrehozása

DESCEND(<kifejezés>) --> komplemens érték

<kifejezés> — tetszőleges karakteres, dátum, logikai vagy numerikus kifejezés. A memo típusú a karakteres típusúnak felel meg.

Visszaadott érték: ugyanolyan típusú kifejezés (kivéve a dátumot, amely numerikus értéket ad vissza), mint a <kifejezés>; annak komplemens értéke. A CHR(0) komplemens értéke mindig CHR(0).

Helye: EXTEND.LIB

Lásd még: FIND, INDEX, SEEK

DEVICEPOS()

A nyomtatófej állítása

DEVICEPOS(<sor>, <oszlop>) --> NIL

<sor> és <oszlop> — a nyomtatófej állásának új sor és oszlop értéke.

Visszaadott érték: mindig NIL.

Lásd még: PCOL(), PROW(), SETPOS(), @...SAY, SET DEVICE, SET PRINTER

DEVOUT()

A kifejezés értékének felírása az aktuális meghajtóra

DEVOUT(<kifejezés>) --> NIL

<kifejezés> — tetszőleges típusú kifejezés.

Visszaadott érték: mindig NIL.

DIRECTORY()

A könyvtár- és állományinformációk tömbbe helyezése

DIRECTORY(<DIR_specifikáció>, [<attributumlista>]) --> tömb

<DIR_specifikáció> — a kívánt meghajtó, könyvtár és állomány megadása. A csoportképző jelek (*, ?) alkalmazhatók. Ha nem adjuk meg, *.* az érvényes.

<attributumlista> — annak meghatározása, hogy a normál állományok mellett milyen attributummal rendelkezők jelenjenek meg a tömbben. Az <attributumlista> az alábbi táblázat szerinti egy vagy több értéket tartalmazhatja:

Attributum	Jelentés
H	Rejtett állományok is
S	Rendszer (system) állományok is
D	Könyvtárak tartalma is
V	Csak a lemeznév megkeresése

Visszaadott érték: egy altömböket tartalmazó tömb, amelyben minden altömb a könyvtár egy állományával kapcsolatos információkat tartalmazza. Az altömbök szerkezete:

Pozíció	Metajelölés	Jelentés	Directry.ch
1	cName	Állomány neve	F_NAME
2	nSize	Mérete	F_SIZE
3	dDate	Felírás dátuma	F_DATE
4	cTime	Felírás ideje	F_TIME
5	cAttributes	Attributum	F_ATT

Helye: CLIPPER.LIB, kapcsolódó állomány Directry.ch

Lásd még: AEVAL(), CURDIR()

DISKSPACE()

A lemezen levő üres hely méretének kiírása

DISKSPACE([<meghajtósorszám>]) --> byte-ok száma

<meghajtósorszám> — egyezményesen 1 az A, 2 a B, 3 a C stb. meghajtó sorszáma. Ha nincs megadva, vagy 0, akkor az aktuális DOS meghajtó, mint alapértelmezés lép érvénybe.

Visszaadott érték: a megadott lemezmeghajtón található üres helyek mérete numerikus egész számként byte-okban.

Helye: EXTEND.LIB, forrásnyelven: EXAMPLEC.C

Lásd még: LASTREC(), LUPDATE(), RECSIZE()

DISPOUT()

A kifejezés megjelenítése a képernyőn

DISPOUT(<kifejezés>) --> NIL

<kifejezés> — tetszőleges típusú kifejezés.

Visszaadott érték: mindig NIL.

DOSERROR()

A legutóbbi DOS hibajelzés kódja

DOSERROR() --> hibakód

Visszaadott érték: numerikus egész számként a legutolsó DOS hibajelzés kódját adja vissza.

Lásd még: ERRORBLOCK(), FERROR()

DOW()

A dátum alapján a nap sorszáma a héten belül

DOW(<dátum>) --> nap sorszáma

<dátum> — a vizsgált dátum.

Visszaadott érték: numerikus, megmondja, hogy a megadott <dátum> a hét hányadik napjára esik (vasárnapot számolja 1. napnak).

Lásd még: CDOW(), DATE(), DAY()

DTOC()

Dátum karaktersorozattá alakítása

DTOC(<dátum>) --> karakteres dátum

<dátum> — a konvertálni kívánt dátum.

Visszaadott érték: a dátum karakteres megfelelője. Alakja az érvényben levő dátumforma szerinti. Az alapértelmezés szerinti forma: hh/nn/éé. Az üres dátum az aktuális dátumformának megfelelő számú üres helyet ad vissza.

Lásd még: CTOD(), DATE(), DTOS(), TRANSFORM(), SET CENTURY, SET DATE

DTOS()

Dátum *éééehhnn* formában karakteresre konvertálása

DTOS(<dátum> --> karakteres dátum

<dátum> — a konvertálandó dátum.

Visszaadott érték: nyolc karakter hosszan, *éééehhnn* formában a dátum. Ha a megadott dátum üres, akkor nyolc szóközt ad vissza. A konvertálás eredménye független az érvényes dátumalaktól.

Lásd még: CTOD(), DATE(), DTOC(), INDEX

EMPTY()

Üres kifejezés viszgálata

EMPTY(<kifejezés>) --> üres-e

<kifejezés> — tetszőleges típusú kifejezés.

Visszaadott érték: igaz (.T.), ha a megadott <kifejezés> üres, és hamis (.F.), ha nem.

Az egyes adattípusok üres értéke

Adattípus	Üres érték
Tömb	Nulla-hossz
Karakter	Szóközök, tabulátorjelek, CR/LF, ""
Numerikus	0
Dátum	Üres (CTOD(""))
Logikai	Hamis (.F.)
Memo	A karakteressel megegyező
NIL	NIL

Lásd még: LEN()

EOF()

Az állomány-vége jel elérésének ellenőrzése

EOF() --> állományvég-e

Visszaadott érték: igaz (.T.), ha a rekordmutató az állomány utolsó rekordja utánra mutat, egyébként hamis (.F.). Ha az aktív munkaterületen nincs nyitott adatbázis, akkor értéke hamis (.F.). Ha az adatbázisnak nincs egyetlen rekordja sem, akkor igaz (.T.).

Lásd még: BOF(), FOUND(), LASTREC(), RECNO(), DO WHILE, GO, LOCATE, SEEK, SKIP

ERRORBLOCK()

Hiba esetén az azt lekezelő kódblokkot tartalmazza

ERRORBLOCK([<hibajelző blokk>]) --> aktuális hibakezelő blokk

<hibajelző blokk> — egy menetközbeni hibánál kerül feltöltésre. Értékét, mint argumentumot a rendszertől kapja.

Visszaadott érték: az aktuális hibakezelő kódblokkot adja vissza. Ha a program futása alatt nem kapott értéket, akkor az aktuális tartalmát adja vissza.

Lásd még: BEGIN SEQUENCE

ERRORLEVEL()

A Clipper visszatérési kód

ERRORLEVEL([<visszatérési kód>]) --> új visszatérési kód

<visszatérési kód> — az új visszatérési kód, amely 0 és 255 közötti numerikus érték lehet. Az alapértelmezés induláskor 0. Ha nem adunk meg argumentumot, az ERRORLEVEL() az aktuális kódértéket mutatja.

Visszaadott érték: Az aktuális *Clipper* visszatérési kód értéke numerikusan, illetve 0.

Lásd még: QUIT

EVAL()

Értékadás egy kódblokk számára

EVAL(<kódblokk>, [<blokkargumentum-lista>]) --> utolsó blokkérték

<kódblokk> — az értékadásra váró kódblokk.

<blokkargumentum-lista> — az értékadás előtt a kódblokknak átadott argumentumlista.

Visszaadott érték: a kódblokk utolsó kifejezésének értéke. Tetszőleges adattípus lehet.

Lásd még: AEVAL(), ASORT(), ASCAN(), DBEVAL()

EXP()

Exponenciális függvény számítása

EXP(<kitevő>) --> természetes logaritmus

<kitevő> — egy logaritmusérték, amelyből egy numerikus értéket kell számolni.

Visszaadott érték: numerikus érték, amely e-nek (mint a természetes logaritmus alapszámának) a <kitevő>-re emelt hatványával egyenlő.

Lásd még: LOG(), SET DECIMALS, SET FIXED

FCLOSE()

Egy nyitott bináris állomány lezárása és a DOS puffer tartalmának lemezre írása

FCLOSE(<azonosító>) --> hibajelzés

<azonosító> — egy előző FOPEN() vagy FCREATE() által adott numerikus érték.

Visszaadott érték: hamis (.F.), ha írás közben egy hiba lépett fel, egyébként igaz (.T.).

Lásd még: FCREATE(), FERROR(), FOPEN(), FREAD(), FREADSTR(), FSEEK(), FWRITE()

FCOUNT()

Az aktuális adatbázis mezőinek száma

FCOUNT() --> mezők száma

Visszaadott érték: az aktuális munkaterületen megnyitott adatbázis mezőinek száma, numerikus egész értékként. Ha nincs nyitott adatbázis, értéke 0.

Lásd még: AFIELDS(), FIELDS(), TYPE()

FCREATE()

Egy nulla hosszúságú bináris lemezállomány létrehozása

FCREATE(<állománynév>, [<attributumszám>]) --> azonosító

<állománynév> — a létrehozandó állomány neve. Ha már van ilyen, azt figyelmeztetés nélkül nulla hosszúságúvá írja át!

<attributumszám> — a táblázatban megadott számok valamelyike. Alapértelmezése 0.

Érték	Fileio.ch	Attributum	Létrehozott állomány
0	FC_NORMAL	Normal	Normál
1	FC_READONLY	Read-only	Csak olvasható
2	FC_HIDDEN	Hidden	Rejtett
4	FC_SYSTEM	System	Rendszer

Visszaadott érték: Az FCREATE() egy 0 és 65.535 közötti azonosító számot ad vissza, amit a későbbi lemezhasználathoz tároljunk el! Hiba esetén a visszaadott érték -1, az FERROR()-ban pedig megjelenik a hiba.

Helye: CLIPPER.LIB, kapcsolódó állomány Fileio.ch

Lásd még: FCLOSE(), FERROR(), FOPEN()

FERASE()

Törlés lemezről

FERASE(<állomány neve>) --> eredmény

<állomány neve> — a törölni kívűnt állomány neve. Kiterjesztéssel együtt kell megadni, tartalmazhat meghajtó és könyvtárnevet is.

Visszaadott érték: -1, ha a törlés sikertelen, egyébként 0. Hiba esetén az FERROR() használható a hiba okának feltárására.

Figyelem! Törlés előtt az állományt le kell zárni!

Lásd még: FERROR(), FRENAME(), CLOSE, ERASE, RENAME

FERROR()

Egy lemezművelet végrehajtásának eredményessége

FERROR() --> hibakód

Visszaadott érték: az utolsó állományművelet DOS hibakódja, numerikus egész értékként. Ha nem jelentkezett hiba, értéke 0.

Az FERROR() értékei

Hibakód	Jelentés
0	Végrehajtva
2	Nem találja az állományt
3	Nem találja a könyvtárt
4	Túl sok nyitott állomány
5	Tiltott művelet
6	Hibás azonosítószám
8	Kevés memória
15	Hibás meghajtó-megadás
19	Írásvédett lemez
21	A meghajtó nem üzemkész
23	CRC adathiba
29	Íráshiba
30	Olvasási hiba
32	Hozzáférés megsértése
33	Zárolás megsértése

Lásd még: DOSERROR(), FCLOSE(), FCREATE(), FOPEN(), FREAD(), FREADSTR(), FWRITE()

FIELD()

Az aktuális adatbázis mezőnevének lekérdezése

FIELD(<mezősorszám>) --> mezőnév

<mezősorszám> — a mező helye az adatbázis szerkezetben.

Visszaadott érték: a megadott mező neve nagybetűs karakteres értékként. Ha nincs olyan sorszámú nező, amelyet kérdezünk, a visszaadott érték 0.

Lásd még: AFIELDS(), DBSTRUCT(), FCOUNT(), LASTREC(), LEN(). TYPE(), VALTYPE(), COPY STRUCTURE EXTENDED

FIELDBLOCK()

Mezőérték kódblokkba helyezése

FIELDBLOCK(<mezőnév>) --> kódblokk

<mezőnév> — annak a mezőnek a neve, amelynek értékét egy kódblokkba helyezzük.

Visszaadott érték: a mező értékét tartalmazó kódblokk.

FIELDGET()

Mezőérték behozatala adatbázisból

FIELDGET(<mezősorszám>) --> mezőérték

<mezősorszám> — numerikus, az aktuális adatbázis struktúrájában elfoglalt helye annak a mezőnek, amelynek értékére szükségünk van.

Visszaadott érték: a behozott mező értéke.

FIELDPUT()

Érték adása egy adatbázismezőnek

FIELDPUT(<mezősorszám>, <kifejezés>) --> mezőérték

<mezősorszám> — numerikus, a mezőnek az adatbázis struktúrájában elfgoglalt helye.

<kifejezés> — ennek értéke kerül a mezőbe beírásra.

Visszaadott érték: A <kifejezés> értéke.

FIELDWBLOCK()

Mező kódblokkba helyezése a megadott munkaterületről

FIELDWBLOCK(<mezőnév>, <munkaterület száma>) --> kódblokk

<mezőnév> — karakteres, annak a mezőnek a neve, amelynek értékét a kódblokkba kívánjuk helyezni.

<munkaterület száma> — a mezőt tartalmazó adatbázis helye (a munkaterület sorszáma).

Visszaadott érték: a mező értékét tartalmazó kódblokk.

FILE()

Egy állomány létezésének ellenőrzése

FILE(<állománynév>) --> van-e

<állománynév> — annak az állománynak a neve, amelynek létezését a Clipper aktuális meghajtóján, illetve könyvtárában meg akarjuk vizsgálni. Használhatók a csoportképző (* és ?) jelek is. Megadható meghajtó és könyvtárnév is.

Visszaadott érték: igaz (.T.), ha létezik a könyvtárban a keresett állomány, egyébként hamis (.F.).

Lásd még: SET DEFAULT, SET PATH

FKLABEL()*

A funkcióbillentyű nevének lekérdezése

FKLABEL(<funkcióbillentyű száma>) --> neve

<funkcióbillentyű száma> — numerikus érték 1 és 40 között.

Visszaadott érték: karaktersorozat, amely a kérdezett funkcióbillentyű nevét (F1, F2 stb.) adja vissza. Ha a <funkcióbillentyű száma> 1-nél kisebb, vagy 40-nél nagyobb, akkor értéke üres karaktersorozat ("").

Helye: EXTEND.LIB, forrásnyelven Examplep.prg

Lásd még: SET FUNCTION

FKMAX()*

A funkcióbillentyűk száma

FKMAX() --> funkcióbillentyűk száma

Visszaadott érték: konstans módon 40-et ad vissza.

Helye: EXTEND.LIB, forrásnyelven Examplep.prg

Lásd még: SET FUNCTION, #define

FLOCK()

Közös használatú adatbázis zárolása (hálózatban)

FLOCK() --> zárolás sikeressége

Visszaadott érték: igaz (.T.), ha az aktuális munkaterületen megnyitott adatbázis zárolása sikeres, egyébként hamis (.F.).

Lásd még: RLOCK(), SET EXCLUSIVE, UNLOCK, USE

FOPEN()

Bináris állomány megnyitása

FOPEN(<állománynév>, [<megnyitási mód>]) --> azonosító szám

<állománynév> — a megnyitni kívánt állomány neve, beleértve a szükséges meghajtó és könyvtár jelölést is.

<megnyitási mód> — numerikus érték, alapértelmezése 0. Értékei a táblázat szerintiek lehetnek:

Mód	Fileio.ch	Művelet
0	FO_READ	Megnyitás olvasásra
1	FO_WRITE	Megnyitás írásra
2	FO_READWRITE	Megnyitás olvasásra és írásra

Visszaadott érték: egy numerikus egész érték 0 és 65535 között, amely a megnyitott állományt azonosítja. Ezt a számot tároljuk el, mert az állomány lezárása csak ennek ismeretében lehetséges! Hiba esetén a visszaadott érték -1.

Helye: CLIPPER.LIB, kapcsolódó állomány Fileio.ch

Lásd még: FCLOSE(), FCREATE(), FERROR()

FOUND()

Az előző keresés sikerességének ellenőrzése

FOUND() --> sikeresség

Visszaadott érték: igaz (.T.), ha az előző parancsban kiadott keresés sikeres volt, egyébként hamis (.F.).

Keresési parancsok: FIND, LOCATE, CONTINUE, SEEK, SET RELATION.

Lásd még: EOF(), CONTINUE, LOCATE, SEEK, SET RELATION, SET SOFTSEEK

FREAD()

Karakterek olvasása bináris állományból puffer változóba

FREAD(<azonosító szám>, @<pufferváltozó>, <byte-szám>) --> byte-ok száma

<azonosító szám> — az FOPEN() vagy az FCREATE() során kapott állományazonosító.

<pufferváltozó> — egy létező és értékkel rendelkező karakteres változó, amelybe az állományból az adatot töltjük. Hosszának nagyobbnak, vagy egyenlőnek kell lenni a <byte-szám>-nál. A változó nevét közvetlenül egy @-jelnek kell megelőznie.

<byte-szám> — a pufferbe beolvasni kívánt byte-ok száma. Az olvasás az aktuális DOS file-pointertől kezdődik.

Visszaadott érték: numerikus egész érték, amely a beolvasott byte-ok számával egyenlő. Ha ez kisebb a <byte-szám>-nál, vagy 0, akkor állomány-vég jellel találkozott, vagy olvasási hiba lépett fel.

Lásd még: BIN2I(), BIN2L(), BIN2W(), FCLOSE(), FCREATE(), FERROR(), FOPEN(), FREADSTR(), FSEEK(), FWRITE()

FREADSTR()

Karakterek olvasása bináris állományból

FREADSTR(<azonosító>, <byte-szám>) --> karaktersorozat

<azonosító> — az FOPEN(), vagy az FCREATE() után kapott azonosító szám.

<byte-szám> — az aktuális DOS file-pointertől beolvasni kívánt byte-ok száma.

Visszaadott érték: legfeljebb 65535 (64K) byte hosszú karaktersorozat. Az üres visszaadott érték ("") vagy állomány-véget, vagy hibát jelent.

Lásd még: BIN2I(), BIN2L(), BIN2W(), FERROR(), FOPEN(), FREAD(), FSEEK(), FWRITE()

FRENAME()

Állomány átnevezése

FRENAME(<régi név>, <új név>) --> NIL

<régi név> — az átnevezni kívánt állomány neve, beleértve a kiterjesztést is. Meghajtó és könyvtár azonosító is megadható.

<új név> — az átnevezés utáni név, beleértve a kiterjesztést is. Meghajtó és könyvtár azonosító is megadható.

Visszaadott érték: -1, ha a művelet sikertelen; és 0 ha sikeres. Az esetleges hiba okát az FERROR() függvénnyel lekérdezhetjük.

Lásd még: FERASE(), FERROR(), FILE(), CLOSE, ERASE, RENAME

FSEEK()

A bináris file-pointert új pozícióba helyezi

FSEEK(<azonosító>, <eltolás>, [<kezdőpozíció>]) --> új pozíció

<azonosító> — az FOPEN() vagy az FCREATE() által adott azonosító szám.

<eltolás> — azoknak a byte-oknak a száma, amennyivel a file-pointert a <kezdőpozíció>-tól eltolni kívánjuk.

<kezdőpozíció> — a file-pointer FSEEK() előtti helyzete. Alapértelmezése 0, ami a BOF-et (állománykezdet) jelzi. Ha a file-pointer az állomány-végen áll, akkor értéke 0 vagy negatív. Ha <kezdőpozíció>-nak 1-et adunk, akkor a jelenlegi helytől folytatja a keresést.

Visszaadott érték: numerikus egész szám, a file-pointer új helyzete az állomány kezdetéhez (0 pozíció) viszonyítva.

Helye: CLIPPER.LIB, kapcsolódó állomány Fileio.ch

Lásd még: FCLOSE(), FCREATE(), FERROR(), FOPEN(), FREAD(), FREADSTR(), FWRITE

FWRITE()

Egy nyitott bináris állományba írás

FWRITE(<azonosító>, <puffer>, [<byte-szám>]) --> felírt byte-ok

<azonosító> — az FOPEN() vagy FCREATE() által adott azonosító szám.

<puffer> — a felírni kívánt karaktersorozat.

<byte-szám> — a file-pointer aktuális helyzetétől kezdődően felírni kívánt byte-ok száma.

Visszaadott érték: numerikus egész érték, a felírt byte-ok száma. Ha ez kisebb, mint a <byte-szám>, vagy 0, akkor a lemez megtelt, vagy egyéb hiba lépett fel.

Lásd még: FCLOSE(), FCREATE(), FERROR(), FOPEN(), I2BIN(), L2BIN()

GETACTIVE()

Az aktív GET objektum behozatala

GETACTIVE() --> GET objektum

Visszaadott érték: Az aktuálisan aktív GET objektum.

GETENV()

A DOS környezeti változó értékének lekérdezése

GETENV(<környezeti változó>) --> karaktersorozat

<környezeti változó> — a DOS környezeti változó neve. A kis- és nagybetűk között különbséget tesz.

Visszaadott érték: karaktersorozatként adja vissza a kért környezeti változó értékét. Ha nincs a kérdezett változó, akkor üres értékkel ("") tér vissza.

HARDCR()

A lágy soremelések kemény soremelésre cserélése

HARDCR(<karaktersorozat>) --> konvertált forma

<karaktersorozat> — a konvertálni kívánt karaktersorozat, vagy memo-mező.

Visszaadott érték: karaktersorozat legfeljebb 65535 (64K) byte hosszan, amelyben a CHR(141) kódok CHR(13)-ra vannak cserélve.

Helye: EXTEND.LIB

Lásd még: MEMOTRAN(), STRTRAN(), ?|??, @...SAZ, EXTERNAL, LABEL FORM, REPORT FORM

HEADER()

Az aktuális adatbázis fejlécrekordjának hossza

HEADER() --> méret

Visszaadott érték: numerikus egész érték, az aktuális adatbázis fejléc rekordjának hossza byte-okban.

Helye: EXTEND.LIB

Lásd még: DISKSPACE(), LASTREC(), RECSIZE()

IF()

Egy feltételre épülő kifejezés eredménye

[I]IF(<feltétel>, <igaz ág>, <hamis ág>) --> érték

<feltétel> — a viszgált logikai kifejezés.

<igaz ág> — ha a feltétel igaz (.T.), akkor ez az érték érvényes.

<hamis ág> — ha a feltétel hamis (.F.), akkor ez az ág érvényes. Nem kell azonos típusúnak lennie, mint az <igaz ág>.

Visszaadott érték: Ha a <feltétel> igaz, akkor az <igaz ág>, ha hamis, akkor a <hamis ág> szerinti érték, az annak megfelelő adattípussal.

Lásd még: DO CASE, IF

INDEXEXT()

Az adatbázis vezérlő meghatározása

INDEXEXT() --> kiterjesztés

Visszaadott érték: karakteres, .NTX, ha Clipper és .NDX, ha dBase III PLus adatbázis vezérlővel dolgozunk.

Lásd még: INDEXKEY(), INDEXORD()

INDEXKEY()

Az index kulcsának meghatározása

INDEXKEY(<sorszám>) --> kulcskifejezés

<sorszám> — az indexlista szerinti sorszám (order-szám). Ha 0-át adunk meg, akkor az aktuális ellenőrző index kulcsát adja vissza.

Visszaadott érték: karaktersorozat, amely a kérdezett kulcskifejezést tartalmazza. Ha nincs a keresettnek megfelelő index, akkor üres értéket ("") ad vissza.

Lásd még: INDEXEXT(), INDEXORD(), INDEX, SET INDEX, SET ORDER, USE

INDEXORD()

Az ellenőrző index helyzetének meghatározása

INDEXORD() --> sorszám

Visszaadott érték: numerikus egész érték, az aktuális munkaterületen megnyitott indexállományok közül az aktuálisnak a sorszáma. A 0 azt jelenti, hogy nincs nyitott indexállomány.

Lásd még: INDEXKEY(), INDEX, SET INDEX, SET ORDER, USE

INKEY()

Egy karakter kivétele a billentyűzetpufferből

INKEY([<másodperc>]) --> INKEY kód

<másodperc> — annak meghatározása, hogy hány másodpercet várjon a program egy billentyű lenyomására. Tizedmásodpercnyi nagyságrend adható meg. Ha 0-át adunk meg, az eltelt időtől függetlenül addig vár, amíg egy billentyűt le nem nyomunk. Ha nem adunk meg semmit, akkor nem vár, és a legutóbb lenyomott billentyű kódját adja vissza.

Visszaadott érték: egy numerikus egész érték -39 és 386 között, amely a billentyűzetpufferből átvett érték INKEY kódja. Ha a billentyűzetpuffer üres, akkor az INKEY() 0 értéket ad vissza. A visszaadott INKEY kód az összes billentyű, funkcióbillentyű, Alt-funkcióbillentyű, Ctrl-funkcióbillentyű, Alt-betű és Ctrl-betű kombinációkat egyaránt tartalmazhatja.

Helye: CLIPPER.LIB, kapcsolódó állomány Inkey.ch

Lásd még: CHR(), LASTKEY(), NEXTKEY(), SET KEY

INT()

A szám egész részének meghatározása

INT(<numerikus kifejezés>) --> egész szám

<numerikus kifejezés> — az egész számra konvertálni kívánt numerikus kifejezés.

Visszaadott érték: a <numerikus kifejezés> egész része. Nem azonos az abszolút értékkel!

Lásd még: ROUND()

ISALPHA()

Egy karaktersorozat betűvel kezdődik-e

ISALPHA(<karaktersorozat>) --> logikai érték

<karaktersorozat> — a vizsgált karaktersorozat

Visszaadott érték: igaz (.T.), ha a vizsgált karaktersorozat betűvel kezdődik (alfabetikus), egyébként hamis (.F.).

Helye: EXTEND.LIB

Lásd még: ISDIGIT(), ISLOWER(), ISUPPER(), LOWER(), UPPER()

ISCOLOR()

A monitortípus meghatározása

ISCOLOR() | ISCOLOUR() --> logikai érték

Visszaadott érték: igaz (.T.), ha a monitor színes, egyébként hamis.

Lásd még: SETCOLOR()

ISDIGIT()

Egy karaktersorozat számmal kezdődik-e

ISDIGIT(<karaktersorozat>) --> logikai érték

<karaktersorozat> — a vizsgált karaktersorozat.

Visszaadott érték: igaz, ha a karaktersorozat első jele 0 és 9 közé eső szám, egyébként hamis.

Helye: EXTEND.LIB

Lásd még: ISALPHA(), ASLOWER(), ISUPPER(), LOWER(), UPPER()

ISLOWER()

Egy karaktersorozat kisbetűvel kezdődik-e

ISLOWER(<karaktersorozat>) --> logikai érték

<karaktersorozat> — a vizsgált karaktersorozat.

Visszaadott érték: igaz (.T.), ha a vizsgált karaktersorozat első jele kisbetű, egyébként hamis.

Lásd még: ISALPHA(), ISDIGIT(), ISUPPER(), LOWER(), UPPER()

ISPRINTER()

Az LPT1 nyomtató készenléti állapotának vizsgálata

ISPRINTER() --> készenlét

Visszaadott érték: igaz (.T.), ha az LPT1 kész állapotban van, egyébként hamis (.F.).

Csak IBM-BIOS kompatibilis rendszerekben működik!

Helye: EXTEND.LIB, forrÁsnyelven EXAMPLEA.ASM

Lásd még: SET DEVICE, SET PRINTER

ISUPPER()

Egy karaktersorozat nagybetűvel kezdődik-e

ISUPPER(<karaktersorozat>) --> logikai érték

<karaktersorozat> — a vizsgált karaktersorozat.

Visszaadott érték: igaz (.T.), ha a vizsgált karaktersorozat első jele nagybetű, egyébként hamis.

Lásd még: ISALPHA(), ISDIGIT(), ISLOWER(), LOWER(), UPPER()

I2BIN()

Egy Clipper numerikus értéket 16-bites bináris egészre konvertál

I2BIN(<egész szám>) --> bináris egész

Visszaadott érték: két byte-os karaktersorozat, amely a 16-bites bináris egész értéket tartalmazza.

Helye: EXTEND.LIB, forrásnyelven Examplea.asm

Lásd még: BIN2I(), BIN2L(), BIN2W, CHR(), FWRITE(), L2BIN()

LASTKEY()

A billentyűzetpuffer utolsó értékének INKEY() kódja

LASTKEY() --> INKEY kód

Visszaadott érték: egy numerikus egész érték -39 és 386 között, amely a billentyűzetpufferbe utolsóként bekerült érték INKEY kódja.

Helye: CLIPPER.LIB, kapcsolódó állomány Inkey.ch

Lásd még: CHR(), INKEY(), NEXTKEY(), KEYBOARD

LASTREC()

A rekordok száma az aktuális adatbázisban

LASTREC() | RECCOUNT()* --> rekordok száma

Visszaadott érték: numerikus egész érték, amely az adatbázis fizikai rekordjainak számát mutatja. A SET FILTER és a SET DELETED nem befolyásolja. Értéke 0, ha nincs nyitott adatbázis.

Lásd még: EOF(), COUNT

LEFT()

A karaktersorozat bal oldali meghatározott része

LEFT(<karaktersorozat>, <hossz>) --> rész karaktersorozat

<karaktersorozat> — amelynek egy részére van szükségünk. Maximális mérete 65535 (64K) byte lehet.

<hossz> — amekkora részre a <karaktersorozat> bal oldaláról szükségünk van.

Visszaadott érték: a <karaktersorozat> bal oldali <hossz> hosszúságú része. Ha a <hossz> negatív, vagy 0, akkor üres érték (""). Ha a <hossz> nagyobb, mint a <karaktersorozat> hossza, akkor azt teljes hosszúságban adja vissza.

Helye: EXTEND.LIB

Lásd még: AT(), LTRIM(), RAT(), RIGHT(), RTRIM(), STUFF(), SUBSTR()

LEN()

A karaktersorozat hossza, illetve a tömb elemeinek száma

LEN(<karaktersorozat> | <tömb>) --> szám

<karaktersorozat> — a vizsgált karaktersorozat.

<tömb> — a vizsgált tömb.

Visszaadott érték: a karaktersorozat hossza, vagy a tömb elemeinek száma numerikus egész értékként. A karaktersorozatnál minden byte-ot egynek számol, beleértve a CHR(0)-t is. Az üres értéket (""), illetve az üres tömb hosszát 0-nak veszi. Egymásbaágyazott tömböknél mindig a legfelső szintnek megfelelő elemszámot adja vissza. Például:

```
LOCAL tomb := { (1,2), (1,2), (1,2)
LEN(tomb)                     // 3
LEN(tomb[1])                  // 2
```

LOG()

Természetes logaritmus számítása

LOG(<numerikus kifejezés>) --> természetes logaritmus

<numerikus kifejezés> — nullánál nagyobb szám.

Visszaadott érték: a szám természetes logaritmus értéke. Ha a <numerikus kifejezés> kisebb vagy egyenlő nullával, akkor numerikus túlcsordulás következik be.

Lásd még: EXP(), SET DECIMALS, SETFIXED

LOWER()

Karaktersorozat kisbetűsre konvertálása

LOWER(<karaktersorozat>) --> kisbetűs alak

<karaktersorozat> — a kisbetűsre konvertálandó karaktersorozat.

Visszaadott érték: a <karaktersorozat> kisbetűs alakja. Csak a betűknél végzi el a konverziót.

Lásd még: ISLOWER(), ISUPPER(), UPPER()

LTRIM()

A baloldali szóközök levágása

LTRIM(<karaktersorozat>) --> balra igazított karaktersorozat

<karaktersorozat> — a balra igazítandó karaktersorozat.

Visszaadott érték: a baloldali vezető szóközökkel csökkentett hosszúságú karaktersorozat.

Lásd még: ALLTRIM(), PAD(), RTRIM(), STR(), SUBSTR()

LUPDATE()

Az utolsó módosítás időpontjának lekérdezése

LUPDATE() --> felírás ideje

Visszaadott érték: az aktuális munkaterületen megnyitott adatbázis utolsó módosításának (felírásának) ideje.

Helye: EXTEND.LIB

Lásd még: FIELD(), LASTREC(), RECSIZE()

L2BIN()

Egy numerikus érték 32-bites bináris egészre konvertálása

L2BIN(<numerikus kifejezés>) --> bináris egész

<numerikus kifejezés> — a konvertálandó érték, a tizedes értékeket figyelmen kívül hagyja.

Visszaadott érték: 4 byte-os karaktersztring, 32-bites bináris egészként értelmezve.

Helye: EXTEND.LIB, forrásnyelven Examplea.asm

Lásd még: BIN2I(), BIN2L(), BIN2W(), CHR(), FWRITE(), I2BIN

MAX()

Két numerikus vagy dátum adat közül a nagyobbik meghatározása

MAX(<num_kif1>, <num_kif2>) --> nagyobb szám
MAX(<dátum_kif1>, <dátum_kif2>) --> nagyobb dátum

<num_kif1> és <num_kif2> — az összehasonlítandó numerikus kifejezések.

<dátum_kif1> és <dátum_kif2> — az összehasonlítandó dátum kifejezések.

Visszaadott érték: a két kifejezés közül a nagyobb érték.

Lásd még: MIN()

MAXCOL()

A maximálisan látható képernyőoszlop száma

MAXCOL() --> oszlop száma

Visszaadott érték: a képernyőn látható jobboldali legszélső oszlop száma.

Lásd még: COL(), MAXROW(), ROW()

MAXROW()

A maximálisan látható képernyősor

MAXROW() --> sor száma

Visszaadott érték: a képernyőn látható legalsó oszlop száma.

Lásd még: COL(), MAXCOL(), ROW()

MEMOEDIT()

Karaktersorozatok és memo mezők megjelenítése és szerkesztése

MEMOEDIT([<karaktersorozat>],
[<felső>], [<bal>], [<alsó>], [<jobb>],
[<szerkesztési mód>], [<saját függvény>],
[<sorhossz>], [<tabulátorméret>],
[<szövegkurzor-sor>], [<szövegkurzor-oszlop>],
[<ablaksor>], [<ablakoszlop>]) --> szövegpuffer

<karaktersorozat> — a MEMOEDIT() szövegpufferébe másolandó karaktersorozat vagy memo-mező. Ha nem adunk meg semmit, a puffer üres.

<felső>, <bal>, <alsó>, <jobb> — az ablak bal felső és jobb alsó sarkát meghatározó koordináták. A sor 0-tól MAXROW(), az oszlop 0-tól MAXCOL() értékig mehet. Alapértelmezés: 0, 0, MAXROW(), MAXCOL().

<szerkesztési mód> — logikai kifejezés, ha igaz (.T.), akkor a szövegpuffer tartalmát szerkeszteni is kívánjuk, ha hamis (.F.), akkor csak megjelenítést kérünk. Alapértelmezés igaz (.T.).

<saját függvény> — egy olyan saját függvény karakteresen (idézőjelek és argumentumok nélkül) megadott neve, amely akkor indul el, ha a MEMOEDIT() által nem kezelt billentyűt nyomunk le, vagy ilyen kódot kap a billentyűzetpufferből.

<sorhossz> — meghatározza a MEMOEDIT() ablakban megjelenített sor hosszát. Ha a megjelenő szöveg ennél hosszabb, akkor a következő sorban folytatódik. Ha az ablak teljes méreténél hosszabb, akkor scrollozást hajt végre. Ha nem adunk meg semmit, akkor a (<jobb> - <bal>) értéket veszi figyelembe.

<tabulátorméret> — a Tab billentyű lenyomására átlépett karakterek száma. Alapértelmezése: 4 szóköz.

<szövegkurzor-sor> és <szövegkurzor-oszlop> — a MEMOEDIT() szerinti megjelenítéskor a kurzor kezdőpozíciója. Mindkét paraméter legkisebb lehetséges értéke 0. Ha nem adunk meg semmit, a MEMOEDIT() ablak 1. sorának 0. oszlopa az érvényes.

<ablaksor> és <ablakoszlop> — a MEMOEDIT() ablakban történő szerkesztéskor a kurzor induló pozíciója. A sor és oszlop értéke 0-tól indul. Ha nem adunk meg semmit, a kurzor az ablak 0. sorában és az éppen aktuális kurzoroszlopban lesz.

Visszaadott érték: a szövegpuffer tartalma, ha a szerkesztést a Ctrl-W billentyűvel vagy a karaktersorozat másolását az Esc billentyűvel zárjuk.

Különleges szerkesztő billentyűk

Billentyű	Hatás
Ctrl-Y	Az aktuális sor törlése
Ctrl-T	A kurzortól jobbra álló rész törlése
Ctrl-B	A bekezdés formázása
Ctrl-V/Ins	Beszúrás mód
Ctrl-W	A szerkesztés befejezése mentéssel
Esc	A szerkesztés befejezése mentés nélkül

Helye: EXTEND.LIB, kapcsolódó állományok Memoedit.ch és Inkey.ch

Lásd még: HARDCR(), LASTKEY(), MEMOLINE(), MEMOREAD(), MEMOTRAN(), MEMOWRIT(), MLCOUNT(), SET SCOREBOARD

MEMOLINE()

Sor kiemelése egy karaktersorozatból vagy memo mezőből

MEMOLINE(<karaktersorozat>,
[<sorhossz>], [<sor száma>],
[<tabulátorméret>], [<szótörés>]) --> karaktersor

<karaktersorozat> — az a memo mező vagy karaktersorozat, amelyből egy sort ki akarunk emelni.

<sorhossz> — a soronkénti karakterek száma. Értéke 4 és 254 között lehet. Alapértelmezése: 79.

<sor száma> — a kiemelni kívánt sor száma. Alapértelmezése: 1.

<tabulátorméret> — a Tab billentyűvel átlépett szóközök száma. Alapértelmezése 4. Ha nagyobb, vagy egyenlő értéket adnánk meg a <sorhossz> értékével, automatikusan (<sorhossz> - 1) értéket vesz fel.

<szótörés> — logikai kifejezés. Igaz (.T.), ha a szótörés funkció bekapcsolt állapotban van, hamis (.F.), ha nincs bekapcsolva. Alapértelmezés: igaz (.T.). Bekapcsolt állapotban a sor végén nem veszi figyelembe az eltört szót; a sor elején azonban figyelembe veszi az előző sor végén álló szótöredéket.

Visszaadott érték: a karaktersorozat, vagy memo-mező megadott sorszámú sora. Ha a megadott <sorhossz> nagyobb az ott található hossznál, a különség szóközökkel kiegészül. Ha a megadott sorú szám nem létezik, üres ("") értékkel tér vissza.

Helye: EXTEND.LIB

Lásd még: MEMOEDIT(), MLCOUNT(), MLPOS()

MEMOREAD()

Lemezállomány tartalmának karaktersorozatként történő visszaadása

MEMOREAD(<állománynév>) --> karaktersorozat

<állománynév> — a lemezről beolvasásra kerülő állomány neve. A kiterjesztés megadása kötelező, meghajtó és könyvtár megadható.

Visszaadott érték: egy szövegállomány karaktersorozatként, amelynek maximális mérete 65535 karakter (64K) lehet. Ha az <állománynév> nem létezik, üres ("") értékkel tér vissza.

Helye: EXTEND.LIB

Lásd még: MEMOEDIT(), MEMOWRIT(), REPLACE

MEMORY()

A szabad memóriaterület

MEMORY(<argumentum>) --> kilobyte

<argumentum> — a szabad memóriaterület lekérdezésének paramétere:

Érték	Jelentés
0	az összes szabad adatterület
1	a legnagyobb összefüggő szabad adatterület
2	a RUN parancsok számára rendelkezésre álló terület

Visszaadott érték: numerikus egész érték kilobyte-okban kifejezve.

MEMOTRAN()

A kemény és lágy sorvégjelek módosítása

MEMOTRAN(<karaktersorozat>,
[<keményvégjel>], [<lágyvégjel>]) --> új karaktersorozat

<karaktersorozat> — a módosítani kívánt karaktersorozat vagy memo-mező.

<keményvégjel> — az a karakter, amelyre a kemény sorvégjelet (CHR(13)) cserélni kívánjuk. Alapértelmezése a pontosvessző (;).

<lágyvégjel> — az a karakter, amelyre a lágy sorvégjelet (CHR(141)) cserélni kívánjuk. Ha nem adunk meg semmit, szóközzel helyettesít.

Visszaadott érték: a módosított karaktersorozat, vagy memo-mező.

Helye: EXTEND.LIB

Lásd még: HARDCR(), STRTRAN(), EXTERNAL, REPORT FORM

MEMOWRIT()

Karaktersorozat vagy memo-mező lemezre írása

MEMOWRIT(<állománynév>, <karaktersorozat>) --> sikeresség jelzése

<állománynév> — annak az állománynak a neve, kiterjesztéssel együtt, amelybe a beírást akarjuk végezni. Meghajtó és könyvtárnév is megadható.

<karaktersorozat> — a felírni kívánt karakteres kifejezés vagy memo-mező neve.

Visszaadott érték: igaz (.T.), ha a felírás sikeres, egyébként hamis (.F.).

Helye: EXTEND.LIB

Lásd még: MEMOEDIT(), MEMOREAD()

MEMVARBLOCK()

Memóriaváltozó értékének kódblokkba helyezése

MEMVARBLOCK(<memóriaváltozó neve>) --> kódblokk

Visszaadott érték: egy kódblokk, amely tartalmazza a memóriaváltozó aktuális értékét.

MIN()

Két numerikus vagy dátum érték közül a kisebbik meghatározása

MIN(<num_kif1>, <num_kif2>) --> kisebb szám
MIN(<dátum_kif1>, <dátum_kif2>) --> kisebb dátum

<num_kif1> és <num_kif2> — az összehasonlítandó numerikus kifejezések.

<dátum_kif1> és <dátum_kif2> — az összehasonlítandó dátum kifejezések.

Visszaadott érték: a két kifejezés közül a kisebb érték.

Lásd még: MAX()

MLCOUNT()

Karaktersorozat vagy memo-mező sorainak száma

**MLCOUNT(<karaktersorozat>, [sorhossz>],
[<tabulátorméret>], [<szótörés>]) --> sorok száma**

<karaktersorozat> — a vizsgált karakteres kifejezés vagy memo-mező.

<sorhossz> — a soronkénti karakterek száma, amely 4 és 254 közötti lehet. Alapértelmezése: 79.

<tabulátorméret> — a Tab billentyű hatására kihagyott szóközök száma. Alapértelmezése: 4. Ha értéke a <sorhossz>-nál nagyobb, automatikusan (<sorhossz> — 1) lesz figyelembe véve.

<szótörés> — a szótörés funkció állapota. Igaz (.T.), ha be van állítva, hamis, ha nincs. Alapértelmezés: az igaz (.T.) érték.

Visszaadott érték: a karaktersorozat vagy memo-mező sorainak száma a megadott sorhosszt, tabulátorértéket és a szótörés állapotát figyelembe véve.

Helye: EXTEND.LIB

Lásd még: MEMOLINE(), MEMOTRAN(), MLPOS()

MLPOS()

Egy karaktersorozatban vagy memo mezőben a megadott sorra lépés

**MLPOS(<karaktersorozat>, <sorhossz>,
<sor száma>, [<tabulátorméret>], [<szótörés>]) --> karakterpozíció**

<karaktersorozat> — a vizsgált karakteres kifejezés vagy memo-mező.

<sorhossz> — a soronkénti karakterek száma.

<sor száma> — annak a sornak a száma, amelyre lépni kívánunk.

<tabulátorméret> — a Tab billentyű hatására kihagyott szóközök száma. Alapértelmezése: 4. Ha értéke a <sorhossz>-nál nagyobb, automatikusan (<sorhossz> — 1) lesz figyelembe véve.

<szótörés> — a szótörés funkció állapota. Igaz (.T.), ha be van állítva, hamis, ha nincs. Alapértelmezés: az igaz (.T.) érték.

Visszaadott érték: a megadott sor első karakterének a karaktersorozaton belül elfoglalt pozícióját adja meg numerikus egész érték alakjában. Ha a <sor száma> nagyobb, mint a <karaktersorozat> sorainak száma, akkor a karaktersorozat hosszát adja vissza.

Helye: EXTEND.LIB

Lásd még: MEMOLINE(), MEMOTRAN(), MLCOUNT()

MOD()*

Két szám dBase III PLUS szerinti osztási maradéka

MOD(<osztandó>, <osztó>) --> maradék

Visszaadott érték: az osztás végrehajtása utáni maradék értéke.

A függvényt a Clipper 5.0 a modulo operátorral (%) helyettesíti, csak a kompatibilitás érdekében maradt meg. Az eredményekben tapasztalható eltérések:

Osztandó	Osztó	%	MOD()	dBase MOD()
3	0	hiba	hiba	3
3	-2	1	-1	-1
-3	2	-1	1	1
-3	0	hiba	hiba	-3
-1	3	-1	2	2
-2	3	-2	1	1
2	-3	2	-1	-1
1	-3	1	-2	-2

Helye: EXTEND.LIB, forrásnyelven Examplep.prg

MONTH()

A hónap számának meghatározása egy dátumkifejezésből

MONTH(<dátumkifejezés>) --> hónap sorszáma

<dátumkifejezés> — a vizsgált dátum.

Visszaadott érték: numerikus egész érték, a dátumból a hónap sorszáma, értéke 0 és 12 közötti lehet. A 0 értéket üres dátum esetén adja.

Lásd még: CMONTH(), DAY(), DOW(), YEAR()

NETERR()

Rákérdezés a hálózati parancs hibás végrehajtására

NETERR() --> hiba-e

Visszaadott érték: igaz (.T.), ha a USE vagy az APPEND BLANK parancs a hálózatban nem hajtható végre. Alapértelmezése hamis (.F.). Igaz (.T.) érték akkor generálódik, ha az állományt egy másik felhasználó USE EXCLUSIVE módon nyitotta meg, vagy APPEND BLANK esetén a LASTREC+1 rekordot másik felhasználó FLOCK() vagy RLOCK()-kal zárolta.

Lásd még: FLOCK(), RLOCK(), APPEND BLANK, USE

NETNAME()

A munkaállomás azonosítójának neve

NETNAME() --> munkaállomás neve

Visszaadott érték: egy maximum 15 karakter hosszúságú karaktersorozat. Ha a munkaállomásnak nincs azonosítója, vagy nem IBM-PC hálózatról van szó, üres ("") értéket ad vissza.

NEXTKEY()

A billentyűzetpuffer következő értéke

NEXTKEY() --> Inkey kód

Visszaadott érték: numerikus egész érték, -39 és 386 között a billentyűzetpuffer következő értékének INKEY kódja. Ha a billentyűzetpuffer üres, vagy a SET TYPEHEAD 0, a 0 értéket adja.

Helye: EXTEND.LIB, kapcsolódó állomány Inkey.ch

Lásd még: INKEY(), LASTKEY(), KEYBOARD, SET TYPEHEAD

OS()

Az operációs rendszer neve

OS() --> operációs rendszer

Visszaadott érték: karaktersorozatként az operációs rendszer neve.

Lásd még: GETENV(), VERSION()

OUTERR()

Értékek átadása a szabvány hibavezérlőnek

OUTERR(<kifejezéslista>) --> NIL

<kifejezéslista> — az átadandó értékek, tetszőleges adattípus lehet.

OUTSTD()

Értékek átadása a szabvány output meghajtónak

OUTSTD(<kifejezéslista>) --> NIL

PAD()

Kifejezés igazítása megadott hosszon belül

PADL(<kifejezés>, <hossz>,
[<feltöltő karakter>]) --> kiegészített karaktersorozat
PADC(<kifejezés>, <hossz>,
[<feltöltő karakter>]) --> kiegészített karaktersorozat
PADR(<kifejezés>, <hossz>,
[<feltöltő karakter>]) --> kiegészített karaktersorozat

<kifejezés> — az igazítandó karakteres, numerikus vagy dátum kifejezés.

<hossz> — a kívánt karaktersorozat hossza.

<feltöltő karakter> — amellyel az igazítás után az üres helyek feltöltésre kerülnek. Alapértelmezés: a szóköz karakter.

Visszaadott érték: mindig karakteres kifejezés. A PADC() a megadott hosszon középre, a PADL() balra, a PADR() jobbra igazítja a kifejezést, a szélső, illetve jobb- vagy baloldali üres helyeket a megadott feltöltő karakterrel egészíti ki.

Helye: EXTEND.LIB

Lásd még: ALLTRIM(), LTRIM(), RTRIM()

PCOL()

A nyomtatófej aktuális oszlophelyzete

PCOL() --> oszlop száma

Visszaadott érték: numerikus egész érték, amely az utolsó kinyomtatott karakternek megfelelő oszlop utáni értéket adja. A kezdő nyomtatási oszloppozíció 0. A SET MARGIN értéket beszámítja.

Lásd még: COL(), PAD(), PROW(), ROW(), SETPRC(), TRANSFORM(), ?|??, @...SAY, EJECT, SET DEVICE, SET PRINTER

PCOUNT()

Az utolsóként átadott paraméter pozíciója

PCOUNT() --> pozíció

Visszaadott érték: numerikus egész érték, amely megmutatja, hogy egy argumentumlistából utoljára hányadik paraméter lett átadva. Ha egy sem, akkor nullát ad.

Lásd még: VALTYPE(), DO, FUNCTION, PARAMETERS, PROCEDURE

PROCLINE()

Az eddig, illetve az előző rutinok által végrehajtott forrássorok száma

PROCLINE([<aktiválás száma>]) --> forrássorok száma

<aktiválás száma> — az aktuálistól visszafelé számítva meghívott rutin sorszáma. Ha 0, vagy nem adunk meg semmit, akkor az aktuális rutinról van szó; egyébként az előző rutinnál 1-et, az azt hívóhoz 2-t stb. kell megadni. Alapértelmezése 0.

Visszaadott érték: Az eddig, illetve a megadott rutin befejező soráig végrehajtott forrássorok száma.

Lásd még: PROCNAME()

PROCNAME()

Az aktuális, vagy az előzőleg végrehajtott rutinok neve

PROCNAME([<aktiválás száma>]) --> eljárásnév

<aktiválás száma> — az aktuálistól visszafelé számítva meghívott rutin sorszáma. Ha 0, vagy nem adunk meg semmit, akkor az aktuális rutinról van szó; egyébként az előző rutinnál 1-et, az azt hívóhoz 2-t stb. kell megadni. Alapértelmezése 0.

Visszaadott érték: az aktuális, vagy a megadott számmal korábban végrehajtott eljárás, függvény vagy kódblokk neve karaktersorozatos alakban.

Lásd még: PROCLINE()

PROW()

A nyomtatófej aktuális sorpozíciója

PROW() --> sor száma

Visszaadott érték: numerikus egész szám, amely a nyomtatófej aktuális sorpozícióját adja. A kezdő sor száma 0.

Lásd még: COL(), PCOL(), ROW(), SETPRC(), ?|??, @...SAY, EJECT, SET DEVICE, SET PRINTER

QOUT()|QQOUT()

Kifejezéslista képernyőn megjelenítése

QOUT([<kifejezéslista>]) --> NIL
QQOUT([<kifejezéslista>]) --> NIL

<kifejezéslista> — tetszőleges típusú adatok vesszővel elválasztott listája (a kódblokk és tömb kivételével), amelyet a képernyőn meg kívánunk jeleníteni. Átirányítható a nyomtatóra. A ?|?? parancsokkal azonos hatású, de azoknál bővebb lehetőségekkel rendelkezik.

Visszaadott érték: mindig NIL. A ROW() és a COL() (illetve a PROW() és a PCOL() értékei aktuálisan változnak.

Lásd még: ?|??, @...SAY, SET ALTERNATE, SET CONSOLE, SET PRINTER, TEXT

RAT()

Egy részkaraktersorozat utolsó előfordulása egy másikban

RAT(<mi>, <miben>) --> kezdőpozíció

<mi> — az a karaktersorozat, amelynek utolsó előfordulását keressük.

<miben> — amelyben a <mi> utolsó előfordulását keressük.

Visszaadott érték: numerikus egész szám, a <mi> jobbról számított első (előlről utolsó) előfordulása a <miben>. Ha nem fordul elő benne, értéke 0.

Helye: EXTEND.LIB

Lásd még: AT(), LEFT(), RIGHT(), STRTRAN(), SUBSTR()

READEXIT()

A fel- illetve lefelé mutató nyílakkal a GET-ből kilépés

READEXIT([<logikai kifejezés>]) --> aktuális helyzet

<logikai kifejezés> — igaz (.T.), ha a le- és felfelé mutató nyílakat a READ-ból kilépésre definiáljuk, hamis (.F.), ha nem. Alapértelmezés hamis (.F.). Ha igazra állítjuk, egy GET-lista első eleméről a felfelé, utolsó eleméről a lefelé mutató nyíllal úgy léphetünk ki, hogy a GET-lista törlődik.

Visszaadott érték: logikai értékként a beállítás helyzete.

Lásd még: READINSERT(), @...GET, READ

READINSERT()

Beszúrásmód beállítása READ-nél és MEMOEDIT-nél

READINSERT([<logikai kifejezés>]) --> aktuális állapot

<logikai kifejezés> — igaz (.T.) esetén a READ és a MEMOEDIT() beszúrás (insert) módban működik, míg ellenkező esetben felülírásmódban. Alapértelmezés hamis (.F.), illetve a felhasználó által utoljára használt mód.

Visszaadott érték: logikai értékként a beszúrásmód beállítása igaz-e vagy sem.

Lásd még: MEMOEDIT(), READEXIT(), READ

READKEY()*

A READ-et megszakító billentyű

READKEY() --> megszakító kód

Visszaadott érték: READ befejezését okozó kód:

Billentyű	Kód
felfelé nyíl	5
lefelé nyíl	2
PgUp	6
PgDn	7
Ctrl-PgUp	31
Ctrl-PgDn	30
Esc	12
Ctrl-End, Ctrl-W	14
Return	15

Helye: EXTEND.LIB

Lásd még: LASTKEY(), NEXTKEY(), READEXIT(), UPDATED(), @...GET, READ

READMODAL()

A GET-lista teljes képernyős szerkesztési módjának aktiválása

READMODAL(<GET-lista tömb>) --> NIL

<GET-lista tömb> — a szerkesztés alatt álló GET objektumok listáját tartalmazó tömb.

Helye: CLIPPER.LIB, forrásnyelven Getsys.prg

Lásd még: @...GET, READ

READVAR()

Az aktuális GET/MENU változó neve

READVAR() --> változónév

Visszaadott érték: nagybetűs karaktersorozatként visszaadja az aktuális GET objektum éppen soron következ változójának, vagy az aktuális MENU TO parancs szerinti változónak a nevét.

Lásd még: @...GET, MENU TO, READ, SET KEY

RECCOUNT()*

Az aktuális adatbázis rekordjainak száma

RECCOUNT() | LASTREC() --> rekordok száma

Visszaadott érték: numerikus egész érték, az aktuális adatbázis fizikai rekordjainak száma. A SET FILTER és a SET DELETED szűrőket figyelmen kívül hagyja. Ha nincs nyitott adatbázis a munkaterületen, értéke 0.

Lásd még: EOF(), LASTREC(), COUNT

RECNO()

A munkaterület aktuális rekordjának sorszáma

RECNO() --> rekordsorszám

Visszaadott érték: numerikus egész érték, a rekord fizikai sorszáma. Ha az adatbázis üres, akkor 1-et ad vissza, de a BOF() és EOF() értéke egyaránt igaz (.T.) és a LASTREC() értéke 0. Ha SKIP-pel az utolsó utáni rekordra lépünk, értéke LASTREC()+1 és az EOF() igaz (.T.). Ha az első rekord elé lépünk (SKIP -1), akkor a logikailag első rekord sorszámát adja vissza, de a BOF() értéke igaz (.T.).

Lásd még: BOF(), EOF(), LASTREC(), GO, SKIP

RECSIZE()

Az adatbázis egy rekordjának hossza

RECSIZE() --> byte

Visszaadott érték: numerikus értékként az aktuális munkaterületen megnyitott adatbázis rekordhossza. Ha nincs nyitott adatbázis, értéke 0.

Helye: EXTEND.LIB

Lásd még: DISKSPACE(), FIELD(), HEADER(), LASTREC()

REPLICATE()

Karaktersorozat ismétlése megadott számszor

REPLICATE(<karaktersorozat>, <ismétlés száma>) --> ismételt karaktersorozat

<karaktersorozat> — az ismételni kívánt karaktersorozat.

<ismétlés száma> — ahányszor egymás után kívánjuk helyezni a <karaktersorozat>-ot.

Visszaadott érték: maximum 65535 (64K) hosszan a megismételt karaktersorozat. Ha 0-át adunk meg az <ismétlés száma> értékeként, akkor üres ("") értéket ad.

Lásd még: SPACE()

RESTSCREEN()

Elmentett képernyőterület megadott helyre visszahelyezése

RESTSCREEN(<felső>, <bal>,
<alsó>, <jobb>, <azonosító>) --> NIL

<felső>, <bal>, <alsó>, <jobb> — a visszatöltésre kerülő képernyőterület elhelyezésének koordinátái.

<azonosító> — karakteres változó, amely egy korábban SAVESCREEN()-nel elmentett képernyőterületet tartalmazza.

Helye: EXTEND.LIB

Lásd még: SAVESCREEN()

RIGHT()

Karaktersorozat jobboldali része

RIGHT(<karaktersorozat>, <hossz>) --> rész kar_sor

<karaktersorozat> — az eredeti karaktersorozat.

<hossz> — a <karaktersorozat> jobb oldaláról ennyi karaktert kívánunk felhasználni.

Visszaadott érték: a <karaktersorozat> jobb oldalának <hossz> darab karaktere. Ha a <hossz> negatív, vagy nulla, akkor üres értéket ("") ad vissza. Ha nagyobb, mint a <karaktersorozat> hossza, akkor a teljes értékét adja. A maximális hossz 65535 (64K) karakter.

Lásd még: LEFT(), LTRIM(), RTRIM(), STUFF(), SUBSTR()

RLOCK()

Az aktív munkaterület aktuális rekordjának zárolása (hálózatban)

RLOCK() --> végrehajtás sikeressége

Visszaadott érték: igaz (.T.), ha a rekord zárolása sikeres, egyébként hamis. Az előzőleg zárolt rekordot feloldja.

Csak hálózatban alkalmazható. Szabály szerint a rekordot módosító parancsoknál kell kiadni. Ezek: @...GET, valamint egy rekordra vonatkozó DELETE, RECALL, REPLACE.

Lásd még: FLOCK(), APPEND BLANK, SET EXCLUSIVE, UNLOCK, USE

ROUND()

Kerekítés megadott tizedesjegyig

ROUND(<kerekítendő>, <tizedesek száma>) --> kerekített érték

<kerekítendő> — a kerekítendő numerikus érték.

<tizedesek száma> — a kerekítés hány tizedesig történjen. Negatív érték esetén az egész számok helyiértékeire vonatkozik.

Visszaadott érték: a kerekített numerikus érték.

Lásd még: INT(), STR(), VAL(), SET DECIMALS, SET FIXED

ROW()

A kurzorpozíció aktuális sora a képernyőn

ROW() --> sor száma

Visszaadott érték: numerikus egész szám, amely megadja a kurzor aktuális sorpozícióját a képernyőn. Értéke 0-tól MAXROW()-ig terjedhet.

Lásd még: COL(), MAXROW(), PCOL(), PROW(), ?|??, @...GET, @...SAY, SET DEVICE

RTRIM()

Jobboldali szóközök levágása

[R]TRIM(<karaktersorozat>) --> rövidített karaktersorozat

<karaktersorozat> — a rövidíteni kívánt karaktersorozat.

Visszaadott érték: a <karaktersorozat> a jobboldali szóközök nélkül. Ha a <karaktersorozat> üres (""), vagy csak szóközökből áll, üres értéket ("") ad vissza.

Lásd még: ALLTRIM(), LTRIM(), PAD(), SUBSTR()

SAVESCREEN()

Képernyő részének elmentése

SAVESCREEN(<felső>, <bal>, <alsó>, <jobb>) --> azonosító

<felső>, <bal>, <alsó>, <jobb> — az elmenteni kívánt képernyőrész koordinátái. Ha az <alsó> és a <jobb> nagyobb a MAXROW(), illetve MAXCOL() értéknél, a képernyő szélét veszi figyelembe.

Visszaadott érték: a képernyő megadott része karaktersorozatként, aminek változóba töltéséről nekünk kell gondoskodni.

Helye: EXTEND.LIB

Lásd még: RESTSCREEN()

SCROLL()

A képernyő görgetése fel vagy le

SCROLL(<felső>, <bal>, <alsó>, <jobb>, <sorok száma>) --> NIL

<felső>, <bal>, <alsó>, <jobb> — a görgetendő terület koordinátái. A sorok és oszlopok a 0, 0, MAXROW(), MAXCOL() értékek között lehetnek.

<sorok száma> — az egyszerre görgetendő sorok száma. Nullánál nagyobb érték a megadott sorral felfelé, kisebb lefelé görget. Nulla érték esetén (alapértelmezés) törli a megadott képernyőterületet.

Helye: EXTEND.LIB

Lásd még: @...BOX, @...CLEAR, @...TO

SECONDS()

Az idő másodpercekben kifejezve

SECONDS() --> másodpercek

Visszaadott érték: a rendszeridő századmásodperc pontossággal másodpercben kifejezve. Értéke 0-tól 86399-ig terjed.

Lásd még: TIME()

SELECT()

A megadott alias-nevű adatbázis munkaterületének száma

SELECT([<aliasnév>]) --> munkaterület száma

<aliasnév> — a keresett munkaterület alias neve. Ha nem adunk meg semmit, az aktuális munkaterületre vonatkozik.

Visszaadott érték: a megadott alias nevű munkaterület sorszáma, numerikus egész érték, amely 0-tól 250-ig terjedhet. A 0 értéket akkor veszi fel, ha nem létezik a megadott alias névvel nyitott munkaterület.

Lásd még: ALIAS(), SELECT, USE

SET()

A SET kapcsolók állapotának lekérdezése/állítása

SET(<azonosító>, [<új érték>]) --> aktuális állapot

<azonosító> — az egyes SET kapcsolók egy-egy azonosító számmal vannak ellátva, amelyeket a Set.ch tartalmaz (ezt #include-dal kapcsolni kell a programhoz). Az egyes <azonosító> értékek konstanshelyettesítőkkel rendelkeznek.

<új érték> — opcionális, a megadott SET kapcsoló módosított értéke. Az <új érték> megadása az eredeti SET kapcsoló által előírt formában történjen.

Visszaadott érték: a megadott SET kapcsoló aktuális értéke. Ha akár az <azonosító>, akár az <új érték> helytelenül lett megadva, a SET() függvény figyelmen kivül marad.

Helye: CLIPPER.LIB, kapcsolódó állomány Set.ch

SETBLINK()

A szín intenzitásának állítása

SETBLINK([<logikai kifejezés>]) --> aktuális fényességszint

<logikai kifejezés> — azt határozza meg, hogy a háttér vagy a kurzor színe intenzívebben világítson, vagy nem. Igaz (.T.) értéknél (ez az alapértelmezés) a kurzor színe, hamis (.F.) értéknél a háttér színe fényesebb.

SETCANCEL()

Az Alt-C megszakítás engedélyezése

SETCANCEL([<logikai kifejezés>]) --> aktuális helyzet

<logikai kifejezés> — igaz (.T.) esetén az Alt-C billentyű lenyomásával leállítható a futó program, hamis (.F.) esetén ez a leállítási lehetőség hatástalan. Alapértelmezés az igaz (.T.).

Visszaadott érték: a beállított érték.

A SET KEY állítás felülírja a SETCANCEL() beállított értékét.

Lásd még: ALTD(), SET ESCAPE, SET KEY

SETCOLOR()

Az aktuális szín átállítása/lekérdezése

SETCOLOR([<színattributumok>]) --> aktuális színek

<színattributumok> — a SET COLOR parancsnál ismertetett attributumok. Értékeik és beállításuk is az ott közöltek szerint történik. Amelyik attributumra itt nem adunk meg értéket, annak az előző értéke marad érvényben.

Visszaadott érték: karaktersorozatként az aktuális színattributum-értékek.

Lásd még: ISCOLOR(), SET COLOR, SET INTENSITY

SETCURSOR()

A kurzor alakjának állítása

SETCURSOR([<alaksorszám>]) --> aktuális kurzoralak

<alaksorszám> — a Setcurs.ch tartalmazza az egyes kurzoralakok sorszámát, amelyek konstanshelyettesítőkkel is rendelkeznek:

Alak	Sorszám	Setcurs.ch
nincs	0	SC_NONE
aláhúzásjel	1	SC_NORMAL
alsó fél blokk	2	SC_INSERT
teljes blokk	3	SC_SPECIAL1
felső fél blokk	4	SC_SPECIAL2

Visszaadott érték: az aktuális kurzoralak numerikus értéke.

Helye: CLIPPER.LIB, kapcsolódó állomány Setcurs.ch

Lásd még: SETPOS(), SET CONSOLE, SET CURSOR

SETKEY()

Műveleti blokk hozzárendelése egy billentyűhöz

SETKEY(<Inkey kód>, [<műveleti blokk>]) --> aktuális művelet

<Inkey kód> — a definiálni kívánt billentyű INKEY() értéke.

<műveleti blokk> — az <Inkey kód> szerinti billentyű lenyomására automatikusan végrehajtandó kódblokk neve. (Csak várakozási állapotban lép érvénybe.)

Visszaadott érték: a végrehajtott művelet, vagy ha nincs a megadott billentyűhöz kódblokk kapcsolva, akkor NIL.

Helye: CLIPPER.LIB, kapcsolódó állomány Inkey.ch

Lásd még: EVAL(), INKEY(), SET KEY

SETMODE()

A képernyőterület méretének módosítása

SETMODE(<sor>, <oszlop>) --> végrehajtás sikeressége

<sor> és <oszlop> — a képernyő megváltoztatott MAXROW() és MAXCOL() értéke.

SETPOS()

A kurzor pozíciójának áthelyezése

SETPOS(<sor>, <oszlop>) --> NIL

<sor> és <oszlop> — a kurzor új pozíciója. Értéke 0, 0 és MAXROW(), MAXCOL() között lehet.

Lásd még: COL(), ROW(), SETCURSOR(), SET CURSOR

SETPRC()

A nyomtatófej aktuális pozíciójának állítása

SETPRC(<sor>, <oszlop>) --> NIL

<sor> és >oszlop> — a nyomtatófej új pozíciója. Ha az új sorérték kisebb a korábbinál, automatikus lapdobás következik be.

Lásd még: PCOL(), PROW(), SET(), SET DEVICE

SOUNDEX()

Karaktersorozat hangzás szerinti alakra (soundex form) konvertálása

SOUNDEX(<karaktersorozat>) --> soundex kód

<karaktersorozat> — a konvertálandó karaktersorozat.

Visszaadott érték: négyjegyű kód A999 maszkalakban.

A soundex-kódról, illetve a hangzás szerinti átalakításról részletesen szó esik a szerző „*Clipper segédprogramok*" című könyvében.

Helye: EXTEND.LIB, forrásnyelven Examplec.c

Lásd még: INDEX, LOCATE, SEEK, SET SOFTSEEK

SPACE()

Karakteres változó szóközökkel feltöltése

SPACE(<hossz>) --> szóközök

<hossz> — a karakteres változó hossza szóközökkel feltöltve.

Visszaadott érték: egy <hossz> hosszúságú karaktersorozat szóközökkel feltöltve. Ha a <hossz> értéke 0, akkor üres érték ("").

Lásd még: PAD(), REPLICATE()

SQRT()

Négyzetgyökvonás pozitív számból

SQRT(<szám>) --> gyökérték

<szám> — egy pozitív szám, amelynek gyökét keressük.

Visszaadott érték: a <szám> gyöke. Ha a <szám> negatív, akkor 0 értéket ad. A gyökérték tizedeseinek számát a SET DECIMALS paranccsal állíthatjuk.

Lásd még: SET DECIMALS, SET FIXED

STR()

Numerikus kifejezés karakteresre alakítása

STR(<szám>, [<hossz>], [<tizedesek>]) --> karakteres szám

<szám> — a konvertálandó numerikus kifejezés.

<hossz> — a konvertálás utáni karaktersorozat hossza. Be kell számítani a tizedespont és a tizedesjegyek számát is.

<tizedesek> — a karaktersorozatba elhelyezendő tizedesjegyek száma.

Visszaadott érték: a <szám> karakteres típusra konvetált alakja a megadott hosszban és megadott számú tizedesjeggyel. Ha a hosszt, illetve a tizedesjegyek számát nem adjuk meg, az alábbi szabályok szerint konvertál:

Kifejezés	Visszaadott hossz
mezőváltozó	mezőhossz tizedesekkel
kifejezés/konstans	minimum 10 számjegy tizedesekkel
VAL()	minimum 3 számjegy
MONTH()/DAY()	3 számjegy
YEAR()	5 számjegy
RECNO()	7 számjegy

Ha a <hossz> kisebb, mint az egész számok jegyeinek száma, akkor a megadott számú csillagot helyez el.

Lásd még: TRANSFORM(), VAL()

STRTRAN()

Egy karaktersorozat vagy memo mező megadott részének cseréje

STRTRAN(<karaktersorozat>, <mit>,
[<mire>], [<honnan>], [<hányszor>]) --> új karaktersorozat

<karaktersorozat> — a módosítandó karaktersorozat vagy memo mező.

<mit> — a <karaktersorozat>-on belül a megváltoztatandó rész.

<mire> — a <mit> rész új értéke. Hosszabb, vagy rövidebb is lehet a <mit>-nél, az új karaktersorozat hossza ennek megfelelően módosul. Ha nincs megadva semmi, akkor az alapértelmezés szerinti üres érték ("") érvényes, ami törli a <karaktersorozat> tartalmát.

<honnan> — a <mit> hányadik előfordulásától legyen az átalakítás végrehajtva. Alapértelmezés: 1.

<hányszor> — a <mit> hány előfordulását alakítsuk át. Alapértelmezés: mindet.

Visszaadott érték: a <karaktersorozat> új értéke esetlegesen megváltozott hosszal.

Lásd még: AT(), HARDCR(), MEMOTRAN(), RAT(), STUFF(), SUBSTR()

STUFF()

Karaktersorozatban karakterek törlése/beszúrása

STUFF(<karaktersorozat>, <honnan>,
<törlendők száma>, <beszúrandó>) --> új karaktersorozat

<karaktersorozat> — a módosítandó karaktersorozat.

<honnan> — a törlés/beszúrás kezdőpozíciója, numerikus érték.

<törlendők száma> — a <honnan> pozíciótól törlendő karakterek száma. Ha 0-át adunk meg, akkor csak beszúrás lesz.

<beszúrandó> — a <honnan> pozíciótól beszúrandó karaktersorozat. Ha üres értéket ("") adunk meg, akkor csak a törlés következik be.

Visszaadott érték: a módosított hosszúságú és tartalmú karaktersorozat.

Helye: EXTEND.LIB, forrásnyelven Examplec.c

Lásd még: AT(), LEFT(), RIGHT(), STRTRAN(), SUBSTR()

SUBSTR()

Rész kivétele egy karaktersorozatból

SUBSTR(<karaktersorozat>, <honnan>, [<mennyit>]) --> rész karaktersorozat

<karaktersorozat> — az eredeti karaktersorozat. Maximális hossza 65535 (64K) karakter lehet.

<honnan> — pozitív érték esetén balról, negatív esetén jobbról számítva az eredeti karaktersorozaton belüli kezdőpozíció.

<mennyit> — megadása esetén azt mutatja, hogy a <honnan> szerinti kezdőpozíciótól hány karaktert veszünk le. Ha nem adjuk meg, vagy nagyobb értéket adunk, mint amennyi karakter azon a részen előfordul, akkor a <karaktersorozat> végéig mindet veszi.

Visszaadott érték: a <karaktersorozat> megadott része.

Lásd még: AT(), LEFT(), RAT(), RIGHT()

TIME()

A rendszeridő lekérdezése

TIME() --> karakteres időérték

Visszaadott érték: a rendszeridő óó:pp:mm alakban karakteresen. Az órát 24 órás formátumban adja.

Lásd még: DATE(), SECONDS(), SUBSTR()

TONE()

Hangelőállítás a megadott frekvencián és időtartamban

TONE(<frekvencia>, <időtartam>) --> NIL

<frekvencia> — a hangmagasságot meghatározó pozitív numerikus érték, hertzben megadva.

<időtartam> — 1/18 másodpercben megadott érték (a 18 érték jelenti az 1 másodpercig tartó hanghatást).

Mindkét értéknél a tizedesjegyek figyelmen kivül maradnak (nem kerekítődnek).

Helye: EXTEND.LIB, forrásnyelven Examplea.asm

Lásd még: CHR(), SET BELL

TRANSFORM()

Tetszőleges adattípus formázott karaktersorozattá alakítása

TRANSFORM(<kifejezés>, <maszk>) --> formázott karaktersorozat

<kifejezés> — tetszőleges Clipper adattípus, kivéve a tömböt, kódblokkot és a NIL értéket.

<maszk> — a @...SAY szerinti PICTURE maszkoknak megfelelő. Idézőjelek közé kell tenni.

Visszaadott érték: a maszk szerint formázott karaktersorozat.

Lásd még: LOWER(), PAD(), STR(), UPPER(), @...SAY

TYPE()

Kifejezés típusának meghatározása

TYPE(<kifejezés>) --> típusjel

<kifejezés> — tetszőleges adattípus karakteresen (idézőjelek között) megadva. Lehet előtaggal, vagy anélkül megadott mezőnév, és függvény is. Idézőjelek nélkül csak akkor adhatjuk meg, ha a változó által hordozott érték karakteres (úgy viselkedik, mint egy makró).

A TYPE() visszaadott értékei

Érték	Jelentés
A	Tömb
B	Blokk
C	Karakteres
D	Dátum
L	Logikai
M	Memo
N	Numerikus
O	Objektum
U	NIL, local vagy static
UE	Szintaktikai hiba
UI	Behelyettesítési hiba

Lásd még: VALTYPE()

UPDATED()

GET objektum változásának lekérdezése

UPDATED() --> változott-e

Visszaadott érték: igaz (.T.), ha az előző READ óta kiadott GET-ek közül valamelyik értéke változott, egyébként hamis (.F.). Ha a GET mezőkön csak végigmegyünk a Return-nel, értéke hamis (.F.). Figyelembe veszi azonban a KEYBOARD paranccsal adott értékeket is, ekkor (.T.).

Lásd még: @...GET, READ, SET KEY

UPPER()

Nagybetűre konvertálás

UPPER(<karaktersorozat>) --> nagybetűs alak

<karaktersorozat> — a módosítandó karaktersorozat.

Visszaadott érték: a <karaktersorozat> nagybetűs alakja. Csak a betűket konvertálja.

Lásd még: ISLOWER(), ISUPPER(), LOWER()

USED()

Egy adatbázis nyitottságának lekérdezése

USED() --> nyitva-e

Visszaadott érték: igaz(.T.), ha az aktuális munkaterületen az adatbázis nyitva van, egyébként hamis (.F.).

Lásd még: ALIAS(), SELECT(), SELECT, USE

VAL()

Karakteres szám numerikus típusra konvertálása

VAL(<karakteres szám>) --> szám

<karakteres szám> — a konvertálandó karakteres kifejezés.

Visszaadott érték: a karakteres típusú szám numerikus alakja. Ha a <karakteres szám> numerikustól eltérő értéket tartalmaz (a baloldali vezető szóközök és a tizedespont kivételével), akkor a visszaadott érték az első nemszám karakterig terjedő szám lesz.

Lásd még: ROUND(), STR(), TRANSFORM(), SET DECIMALS(), SET FIXED

VALTYPE()

Tetszőleges kifejezés adattípusának meghatározása

VALTYPE(<kifejezés>) --> típusjel

<kifejezés> — tetszőleges típusú kifejezés, beleértve a local és staic változókat, felhasználói és EXTEND.LIB függvényeket is.

Visszaadott érték: Egy karakteren jelöli a kifejezés típusát:

Érték	Jelentés
A	Tömb
B	Blokk
C	Karakteres
D	Dátum
L	Logikai
M	Memo
N	Numerikus
O	Objektum
U	NIL

Lásd még: TYPE

VERSION()

A Clipper verzió lekérdezése

VERSION() --> verziószám

Visszaadott érték: karakteresen visszaadja a használt Clipper verzió számát.

WORD()*

Numerikus érték duplapontosságú egészre konvertálása

WORD(<szám>) --> NIL

<szám> — egy legfeljebb 16-bites (-32767 és 32767 közötti) szám duplapontosságú egészre konvertálása egy CALL parancsnak történő értékátadáshoz.

Visszaadott érték: A CALL-nak átadja az értéket, egyébként NIL értékkel tér vissza.

YEAR()

Dátumkifejezésből az év értéke numerikusan

YEAR(<dátum>) --> évszám

<dátum> — a konvertálandó dátum.

Visszaadott érték: a dátumból az évszám numerikusan. Az üres dátum(CTOD("")) 0 értéket ad vissza.

Lásd még: DAY(), MONTH()

A Clipper fordítóprogramja

A fordítóprogram (CLIPPER.EXE) a forrásnyelvű programokból egy átmeneti (.obj) állományt hoz létre. Ezeknek a — tárgykódnak is nevezett — állományoknak az összeszerkesztésével áll elő az a memóriába tölthető modul (.exe), amely a program futtatását biztosítja. Egy programrendszer egy vagy több tárgykódú állomány összeszerkesztésével állítható össze. Ha külön paraméterként nem adunk meg mást,akkor a főprogram nevének megadásával az összes hívatkozott program, eljárás és formátumállomány lefordításra kerül (kivéve azok, amelyeknek a neve csak makróértékként fordul elő). Ha csak szintaktikai ellenőrzést kívánunk végezni (tárgykód létrehozása nélkül), azt a megfelelő paraméterrel jelezhetjük. Lehetőség van arra, hogy egy külön leíró állományban felsoroljuk azoknak a programoknak a neveit, amelyeket egy .obj állományba kívánunk fordítani. Ennek az overlay alkalmazásánál van jelentősége. A különböző lehetőségek részletes ismertetésére a paraméterek tárgyalásánál kerül sor.

A fordítóprogram és környezete

A Clipper fordítóprogram indításának alakja:

CLIPPER [<forrásprogram> | @<leíró állomány>] [<paraméterek>]

<forrásprogram> - a lefordítani kívánt forrásprogram neve. A kiterjesztés alapértelmezése .prg. A név tartalmazhat meghajtó és könyvtárnevet is.

<leíró állomány> - az azonos tárgyállományba lefordítani kívánt forrásprogramok neveit tartalmazó állomány neve. A kiterjesztés alapértelmezése .clp.

<paraméterek> - a fordítóprogram működését vezérlő egy vagy több paraméter (később részletesen ismertetésre kerülnek). Kis- és nagybetűvel egyaránt megadhatók. Minden paramétert a per (/), vagy a kötőjel (-) vezet be.

Ha a CLIPPER szó után nem adunk meg semmit, akkor az indítással kapcsolatos fontosabb tudnivalók jelennek meg a képernyőn.

A környezeti változó

A gyakran használt paraméterek a CLIPPERCMD környezeti változóban definiálhatók. Ezzel megtakaríthatjuk magunknak azt az időt, amit ezeknek az állandó beírása venne el. A CLIPPERCMD környezeti változó az AUTOEXEC.BAT állományba helyezve biztosítja állandó érvényességét. Megadásának formája:

SET CLIPPERCMD=[<paraméterek>]

<paraméterek> - a minden fordításnál alkalmazni kívánt paraméterek, amiket itt ugyanúgy kell megadni, mintha a fordítóprogram indításának parancssorában adtuk volna meg. Az itt megadott paraméterértékek a fordítóprogram indításakor felülírhatók.

Példák:

A sorszámozás elnyomása és új kapcsolódó állomány illesztése:

```
SET CLIPPERCMD=/L /ULisestd.ch
```

A tárgykód állományok elhelyezésére szolgáló könyvtár definiálása:

```
SET CLIPPERCMD=/OC:\CLIPPER5\OBJ\
```

A leíró állomány

A leíró állomány egy egyszerű szövegállomány, amely azoknak a programoknak, eljárásoknak és formátumállományoknak a neveit tartalmazza, amelyeket egy tárgyállományba kívánunk fordítani. A létrejövő tárgyállomány neve — hacsak a /O paraméterrel mást nem írunk elő — a leíró állomány neve lesz. Leíró állomány alkalmazása esetén a hívott programok és eljárások nem kerülnek lefordításra, úgy működik, mintha mindegyik programhoz a /M paramétert adtuk volna meg.

A leíró állomány összeállításának szabályai:

- a forrásprogramok neveit egymástól a CR-LF kódokkal (Return) válaszuk el;
- a forrásprogramok neveit kiterjesztéssel vagy anélkül adhatjuk meg. Ha nem adunk meg kiterjesztést, az alapértelmezés .prg;
- a forrásprogramok neveinél megadhatjuk a meghajtó és a könyvtár nevét is.

A visszatérési kód

Lekezelhetetlen hiba, vagy felhasználó általi megszakítás (Ctrl-C) esetén a DOS visszatérési kódja 1, és ebben az esetben tárgyállomány nem készül. Normális fordítás esetén a visszatérési kód 0. Figyelmeztető üzenet (warning) esetén a fordítás normálisan fejeződik be, a DOS visszatérési kód a DOS ERRORLEVEL kulcsszavával vizsgálható meg.

Hiba esetén a hibajelzés a képernyőn megjelenik, ami átirányítható egy szövegállományba:

C>CLIPPER <forrásprogram> > HIBAK.TXT

Az egyes hibakódok leírását nem ismertetjük, azok már több kiadványban megjelentek, például a szerző „*Clipper segédprogramok*" című könyvében.

Az előfordító működése

A Clipper 5.0 a fordítást két menetben végzi: először az előfordító (pre-processzor) az előfordítást, majd a fordító (compiler) a fordítási fázist hajtja végre.

Az előfordító megvizsgálja a forrásprogramot és a direktívákat szabályos Clipper forráskódra alakítja, majd ennek eredményét átadja a fordítónak. Programunk előfordító utáni állapota a /P paraméter megadásával egy .ppo kiterjesztésű állományba íratható. Ez különösen a #command és a #translate direktívák helyes használatának ellenőrzéséhez hasznos. Az előfordító direktívákat külön fejezetben tárgyaljuk.

A fordítási fázis

A Clipper fordító a forráskódban írt programot „*Intel object code*"-dá alakítja, aminek az .obj kiterjesztést adja. Ebből a tárgykódú állományból készít a szerkesztő program tölthető (.exe kiterjesztésű) állományt. Minden DO, SET PROCEDURE és SET FORMAT program, eljárás és formátumhivatkozás befordításra kerül, kivéve ha a /M paraméterrel ezt felülbíráljuk.

A fordítással kapcsolatos fontosabb tudnivalók:

- először a fordítóprogram parancssorában megadott program kerül lefordításra;
- a meghívott programok, eljárások és formátumok befordítása a hívás sorrendjében történik;

- a DO utasítással meghívott eljárásokat — ha talál ilyen nevű .prg állományt — a fordító az első hívás helyén fordítja be. Ha nem találja, akkor figyelmeztető üzenetet ad. (Vigyázzunk arra, hogy a hívatkozott állományok ugyanabban a könyvtárban legyenek, mint a hívó program, mert a fordító nem keresi máshol ezeket!);
- a SET FORMAT-tal megadott hívatkozással — ha mást nem adunk meg — egy megadott nevű .fmt kiterjesztésű állományt fordít be. Ha ugyanilyen névvel már befordításra került egy eljárás, akkor ezt az eljárást fogja figyelembe venni, és nem az .fmt állományt!

Leíró állomány alkalmazása esetén a programok az ott megadott sorrendben egyenként kerülnek befordításra, függetlenül azok hívatkozásaitól.

A fordítási és szerkesztési parancsok beírása, különösen sok paraméter esetén, hosszadalmas lehet. Ennek gyorsítását (a már említett CLIPPERCMD és a szerkesztő programnál ismertetésre kerülő RTLINKCMD környezeti változók alkalmazása mellett) parancsállomány megírásával érhetjük el. A példaként bemutatott CL.BAT parancsállomány alkalmazásával egy egyszerű

CL <forrásprogram>

beírásával a fordítási és szerkesztési menet is végrehajtásra kerül.

Például a CL.BAT egy lehetséges formája:

```
CLIPPER %1 /L /O:\CLIPPER5\OBJ\ /USajatstd.ch
RTLINK %1 /POSITIONAL
```

A kapcsolódó állományok

A kapcsolódó (header) állományok előfordító direktívákat és parancsdefiníciókat tartalmaznak. A kapcsolódó állományok kiterjesztésének nincs megadott alapértelmezése. Illesztésük az #include direktívával, vagy a fordítóprogram /U paraméterével lehetséges. Az INCLUDE környezeti változó DOS-ban (például az AUTOEXEC.BAT-BAN) történő megadásával biztosíthatjuk a kapcsolódó állományok egy könyvtárba (például a \CLIPPER5\INCLUDE-ba) helyezését, és onnan való elérésüket. Ehhez a következőt kell megadni:

```
SET INCLUDE=<könyvtárnév>
```

Ha az aktuális könyvtárban nincs a hívatkozott kapcsolódó állomány, akkor az esetleges /I paraméterrel megadott könyvtárban, majd a SET INCLUDE környezeti változóval definiált könyvtárban keresi a kapcsolódó állományt.

A fordító által létrehozott állományok

A fordítóprogram három állományt hozhat létre, ezek:

- a tárgyállomány (.obj);
- az ideiglenes állomány(ok) (.tmp);
- az előfordító által előállított forrásprogram (.ppo).

Ha nem bíráljuk felül, akkor ezeket az állományokat az aktuális könyvtárba helyezi, kiterjesztésük — az előfordító által előállított .ppo kivételével — felülírható.

A *tárgyállomány* a normálisan befejeződött fordítás után jön létre. Neve (ha a /O paraméterrel nem írjuk felül) megegyezik a fordítóprogram parancssorában megadott névvel (ami vagy a főprogram, vagy a leíró állomány neve). Kiterjesztése .obj.

Ideiglenes állomány a fordítás folyamán keletkezik, a későbbiekben erre nincs szükség. Ezért célszerű azokat egy könyvtárba átirányítani, és a későbbiek folyamán törölni. Az átirányítás formája:

SET TMP=<meghajtó[\könyvtárnév]>

Az átirányítást különösen akkor célszerű alkalmazni, ha kevés helyünk van a lemezen. Ekkor megtehetjük azt is, hogy az ideiglenes állományokat a memóriában kialakított virtuális lemezre írjuk, például:

```
SET TMP=D:
```

Figyelem! A /T paraméter is használható az ideiglenes állományok átirányítására. Ha ezt megadtuk, akkor a SET TMP környezeti változóban megadott érték hatástalan!

Az előfordító által előállított *forrásprogram* csak a /P paraméter megadása esetén kerül a lemezre. Neve a fordítóprogram parancssorában megadott név (főprogram vagy leíró állomány neve), kiterjesztése .ppo lesz.

A környezeti változók megadásának hatása

Ha az eddig megadott környezeti változók (CLIPPERCMD, INCLUDE, TMP) mindegyikét megadjuk, lehetséges, hogy kevésnek bizonyul a rendszernek erre fentartott területe. A 3.2 DOS verziótól kezdve ez a terület módosítható a CONFIG.SYS állomány SHELL parancsával. Például:

```
SHELL=C:\COMMAND.COM C:\ /P /E:2048
```

megadásával ezt a területet 2048 byte-ra növeljük. Vigyázzunk arra, hogy a /E paraméter megadása esetén — a példában megadott módon — kötelezően meg kell adnunk a COMMAND.COM nevét és helyét!

A fordítóprogram paraméterei

A paraméterekkel írhatjuk elő a fordítóprogram viselkedését. Megadhatók a parancssorban, vagy a CLIPPERCMD környezeti változóval. A parancssorban megadott paraméterek felülírják a CLIPPERCMD-ben definiáltakat.

A paraméterek egyaránt megadhatók kis- vagy nagybetűvel, eléjük kötelezően a per (/) vagy a kötőjelet (-) kell írni. Szóközzel elválasztva tetszés szerinti sorrendben követhetik egymást. Amelyik paraméter argumentummal rendelkezik, annál az argumentumot a paraméter azonosítóbetűje utűn közvetlenül (szóköz hagyása nélkül) kell írni.

/A

A PRIVATE, PUBLIC vagy PARAMETERS utasításban megadott változók automatikusan MEMVAR-ként kerülnek deklarálásra.

/B

A tárgyállományba helyezi a sorszámokat, a local és static változóneveket, a forrásállomány nevét, és más olyan információt, amit a *Debugger* használ. Ha a /L paramétert is megadjuk, akkor a sorszámok nem kerülnek be.

/D<azonosító>[=<szöveg>]

Az előfordító számára definiál egy azonosítót. Ha a <szöveg> nincs megadva, akkor az <azonosító> egy üres értéket vesz fel. Ha a szövegben dupla idézőjelet kívánunk használni, elé a \ jelet kell írni! Ez a paraméter szolgálhat a feltételes fordítás megfelelő ágának végrehajtásához (lásd az #if direktívánál leírtakat).

/I<könyvtárnév>

A paraméter megadása esetén a fordítóprogram a kapcsolódó (header) állományt először ebben a könyvtárban keresi, és csak ezt követően az INCLUDE környezeti változóval definiáltban. Több /I paraméter is megadható egy fordítási parancsban.

/L

Nem helyezi el a forrásprogram sorszámait a tárgyállományban. A paraméter hatására a tárgyállomány mérete soronként 3 byte-tal kevesebb lesz. Alkalmazása csak a hibátlan program fordítása esetén javasolt, mert egyébként a jelzett hibát nem tudjuk behatárolni, illetve a *Clipper Debugger* nyomkövető sem alkalmazható. A /L és a /B paraméterek együttes használata esetén a *Debugger* információk közül a sorszámok fognak csak hiányozni.

/M

Csak az aktuális forrás (.prg) állomány fordítását hajtja végre, külső hívatkozásait nem. A paraméter hatására a DO, SET PROCEDURE és SET FORMAT utáni hívatkozások (hacsak nem az aktuális program részét képezik) külső hívatkozásnak számítanak. Ugyanakkor a #include direktívával illesztett kapcsolódó állományok fordítása végrehajtásra kerül. A /M paraméternek nincs hatása a leíró állományban.

/N

Segítségével elérhetjük azt, hogy egyik eljárás sem lesz elsődleges a többivel szemben. Alkalmazásának célja a teljes hatókörű változók deklarálásának lehetősége.

Ha a /N paraméter nincs megadva, akkor a fordítás után a forrásprogram elejétől az első PROCEDURE vagy FUNCTION utasításszóig terjedő rész számít kezdőeljárásnak. A program hierarchiája ebből az eljárásból indul ki, és ez végig megmarad. Ha a /N paramétert megadjuk, akkor mindig az az eljárás számít a hierarchia legfelső szintjének, amely éppen érvényben van.

/O<tárgyállomány neve>

A lefordított tárgyállományt a megadott néven helyezi el a lemezen. Ha csak meghajtó és könyvtárnevet adunk meg, akkor a megadott könyvtárba kerül. Vigyázzunk arra, hogy ebben az esetben a könyvtár nevét a \ karakterrel zárjuk le, ellenkező esetben a könyvtárnévvel azonos nevű .obj kiterjesztésű állományt hoz létre.

/P

Lemezre írja az előfordító által a fordítóprogramnak átadott forrásprogramot, amelyben az előfordító direktívák szabályos Clipper utasításokként szerepelnek. A létrejövő állomány neve az elsőként megadott forrásprogram neve .ppo kiterjesztéssel. A /O paraméter az állomány elhelyezésére és nevére nincs hatással.

/Q

Hatására a képernyőn nem jelenik meg az éppen fordítás alatt álló sor száma. A tárgyállományba azonban bekerülnek a sorok számai, kivéve, ha a /L paramétert is megadjuk.

/R[<lib_könyvtárnév>]

A szerkesztőprogram számára ezzel a paraméterrel adható meg, hol találhatók a programban nem lekezelt külső hívatkozások. Több /R paraméter is megadható, mindegyik érvényes lesz. A /R paraméter <lib_könyvtárnév> nélkül történő megadása azt jelenti, hogy nincs ilyen könyvtár.

Vigyázat! A Clipper 5.0 a CLIPPER.LIB, EXTEND.LIB és DBFNTX.LIB könyvtárakat alapértelmezésként láncoltnak tekinti, amely hívatkozást a /R önmagában történő megadása törli! A /R után írt <lib_könyvtárnév>-ben felsorolt könyvtárak a fentiekhez láncolódnak.

/S

Lehetővé teszi a program szintaktikai ellenőrzését úgy, hogy nem készít (.obj) tárgyállományt. Ha hozzá még a /M paramétert is megadjuk, akkor csak az aktuális programot fordítja, annak hívatkozásai nélkül.

/T<meghajtó[\könyvtárnév]>

Az ideiglenes állományok elhelyezésére szolgáló könyvtár nevét adhatjuk meg vele. Felülírja a SET TMP környezeti változóban megadott értéket.

/U[<kapcsolódó állomány>]

Egy saját magunk által készített kapcsolódó állomány nevét adhatjuk meg az előfordító számára. Ez az állomány csak előfordító direktívákat tartalmazhat. A kapcsolódó állomány helyének keresése során a Clipper a következő sorrendet követi: aktuális könyvtár, /I paraméternél megadott könyvtár(ak), INCLUDE környezeti változóban megadott könyvtár.

/V

Hatására a nemdeklarált és aliasnévvel el nem látott változók dinamikus memóriaváltozók lesznek, úgy viselkednek, mintha a MEMVAR-> előtaggal lennének ellátva.

/W

Hatására üzenet formájában kiírásra kerülnek a nemdeklarált és alias névvel el nem látott változók. Használata akkor célszerű, amikor *private* és *public* változókat kívánunk *local* vagy *static* formára konvertálni.

Példák:

A Sue.prg programot csak szintaktikailag kívánjuk ellenőrízni, letíltva a fordítás ideje alatt a sorszámok megjelenítését:

```
CLIPPER Sue /S /Q /M
```

A fordítást a PROGOK.CLP leíró állomány alapján végezzük. A Debugger használatához a nyomkövetéshez szükséges információkat a tárgyállományba fordíttatjuk, amelyet a C:\CLIPPER5\OBJ könyvtárban TARGYAL.OBJ néven helyezünk el:

```
CLIPPER @PROGOK /B /OC:\CLIPPER5\OBJ\TARGYAL
```

A PROGOK.CLP tartalma például a következő:

```
PROG1
PROG2
PROG3
PROG4
```

Az előzőekben megadott tudnivalók mellé (amelyek szükségesek egy forrásállomány, vagy egy teljes programrendszer lefordításához), néhány gyakorlati tanács:

- A különböző típusú állományokat — a könnyebb áttekinthetőség érdekében — helyezzük különböző könyvtárakba (a Clipper 5.0 javasolt könyvtárszerkezete szerint).
- Alkalmazzuk a javasolt környezeti változókat! Különösen a leggyakrabban használt paraméterek ide helyezésével takaríthatunk meg sok időt, és szűrhetjük ki az elmaradásukból eredő hibalehetőséget.
- Ne féljünk a Clipper 5.0 nyújtotta új szolgáltatásoktól (előfordító direktívák, kódobjektumok alkalmazása) — ezek használatával sokkal gyorsabb és kényelmesebben kezelhető programokhoz jutunk.
- Nagy programrendszer tesztelése esetén a már jó programokat fordítsuk egy külön tárgyállományba, a még tesztelés alatt állókat pedig vagy külön-külön, vagy egy másikba. Az elkülönítésre a leíró állományok szolgálhatnak. Ezzel jelentős időmegtakarítás érhető el, hiszen a jó programokat már nem kell minden javítás után újra fordítani.
- A leíró állományt az előző eseten kívül csak overlay alkalmazása esetén érdemes használni, magában rejti ugyanis annak a hibának a lehetőségét, hogy valamelyik eljárást kifelejtjük a felsorolásból.
- Overlay technikát csak akkor alkalmazzunk, ha az szükséges!
- Kevés lemezterület esetén az ideiglenes (.tmp) állományokat irányítsuk virtuális lemezre (a memóriába). Egyébként is javasolt elkülönítésük, és időközönkénti törlésük.
- Vigyázzunk arra, hogy a programból ne indítsunk memóriarezidens programot, mert a visszatérés a programunkba nem mindig garantált!
- Programtesztelés időszakában alkalmazzuk a következő fogásokat: 1 - szintaktikai ellenőrzés (/S); 2 - jó programok elkülönítése; 3 - szemantikai hiba feltárásához a Debugger használata.

Az RTLINK szerkesztőprogram

A fordítóprogram által előállított .obj kiterjesztésű tárgymodulokat a memóriába betölthető és futtatható formára kell hozni. Ezt végzi el a szerkesztőprogram. A korábbi Clipper verzióhoz adott PLINK86 lassúságáról volt közismert, ezért inkább a Borland cég TLINK programját alkalmazták, amit viszont overlay technikát alkalmazó Clipper programokra nem lehetett használni.

A Clipper 5.0 verzióhoz adott RTLINK szerkesztőprogram kiküszöböli a PLINK86 hiányosságait, sőt alapvetően új szolgáltatásokkal egészítették ki, amiből az elő- és összehasonlító szerkesztés lehetőségét, valamint azt emeljük ki, hogy az előállításra kerülő .exe állomány a korábbinál jóval kisebb lesz.

Az RTLINK szerkesztőprogram segítségével C, illetve Assembler nyelvről lefordított tárgymodulok is programunkhoz illeszthetők, de ebben az esetben csak a statikus overlay lehetőségét hagyja nyitva.

A szerkesztőprogram és környezete

RTLINK.EXE

Maga a szerkesztő program, ez hozza létre az .exe állományt, valamint az előszerkesztés .pll könyvtárállományát. Indítása a DOS-ból. Kezelését később ismertetjük.

RTLINKST.COM

Ez az állomány tartalmazza az előszerkesztett .pll könyvtárállomány kezdőkódját. Az RTLINKST.COM csak akkor fejti ki hatását, ha a szerkesztőprogram PRELINK paraméterével .pll állományt hozunk létre. Ugyanabban a könyvtárban kell elhelyezkednie, ahol az RTLINK.EXE található.

RTLUTILS.LIB

Ez a könyvtár tartalmazza a statikus overlayvezérlőt. Akkor használjuk, amikor a programot statikus overlay technikával szerkesztjük.

RTLINK.DAT

Tartalmazza az összes RTLINK hibaüzenetet. Ugyanabban a könyvtárban kell elhelyezkednie, ahol az RTLINK.EXE található.

RTLINK.HLP

A HELP állományt tartalmazza, amely akkor tekinthető meg, amikor az RTLINK-et a /HELP paraméterrel indítjuk. Ugyanabban a könyvtárban kell elhelyezkednie, ahol az RTLINK.EXE található.

A felhasználói csatolók

Az RTLINK két felhasználói csatolóval rendelkezik: a **POSITIONAL** a Microsoft LINK, míg a **FREEFORMAT** a Plink86-Plus formátumát követi. A két csatoló alkalmazása közötti legszembetűnőbb különbség a paramétermegadás eltérő volta. A fejlesztők a FREEFORMAT alkalmazását javasolják (ez az alapértelmezés is), amely rugalmasabb és hatékonyabb.

A szerkesztőprogram mindkét csatoló esetén háromféleképpen indítható:

- parancsmódban;
- prompt-módban
- leíró állomány segítségével.

A szerkesztőprogram indítása

a)FREEFORMAT csatolóval

RTLINK [FILE <tárgymodul lista>
[OUTPUT <tölthető modul neve>]
[LIBRARY [<könyvtárlista>]
[<paraméterlista>]]

FILE <tárgymodul lista> - itt kell vesszővel elválasztva megadni az összeszerkeszteni kívánt tárgymodulok és könyvtárállományok neveit. Ha itt adunk meg könyvtárnevet, akkor a könyvtár teljes tartalma beszerkesztésre kerül. A kiterjesztés alapértelmezése: .OBJ. Ha több FILE kulcsszót adunk meg egy parancsban, akkor az összes FILE utáni hivatkozást figyelembe veszi. A FILE kulcsszó helyett elegendő a FI megadása.

OUTPUT <tölthető modul neve> - a létrejövő .EXE állomány nevét határozhatjuk meg. Ha nem adunk meg semmit, akkor a FILE utáni első tárgymodul neve lesz a tölthető modul neve. Ha a PRELINK paramétert alkalmazzuk, akkor az .EXE állományon kívül egy .PLL és egy .PLT állományt is létrehoz. Ha több OUTPUT kulcsszót tartalmaz ugyanaz a parancs, az utolsót veszi figyelembe. Az OUTPUT kulcsszó helyett elegendő az OU megadása.

LIBRARY<könyvtárlista> - vesszővel elválasztva itt kell megadni azokat a könyvtár és/vagy tárgymodulneveket, amelyek az egyébként lekezeletlen hivatkozásokat tartalmazzák. A kiterjesztés alapértelmezése: .LIB. A LIBRARY kulcsszó helyett elegendő a LI megadása. Ha itt adjuk meg a könyvtárat, akkor az .exe állományba csak a ténylegesen használt rutinok kerülnek beszerkesztésre.

Előszerkesztett (.PLL) állományok esetén a .LIB könyvtárak láncolása a .PLL állományhoz nem automatikus, ehhez a REFER paramétert meg kell adni.

Figyelem! A .PLL állomány nevét ne adjuk meg a könyvtárlistában! Erre a célra a PLL paraméter szolgál.

<paraméterlista> - a későbbiekben ismertetésre kerülő paraméterek felsorolása. Ezek a két felhasználói csatolónál megegyeznek, de megadásuk módja eltérő: a POSITIONAL csatolónál a per (/) jel, míg a FREEFORMAT csatolónál a szóköz választja el őket egymástól.

Leíró állomány alkalmazása esetén az indítás formája:

RTLINK @<leíró állomány> <paraméterlista>

<leíróállomány> - egy egyszerű ASCII szövegállomány, amelynek alapértelmezés szerinti kiterjesztése: .LNK. Alkalmazása esetén a szerkesztőprogram az előzőekben ismertetett állománynév megadásokat ebből a szövegállományból veszi. Használata azért célszerű, mert nem kell minden alkalommal a hosszadalmas parancssort beírni, hanem kész leíró állományok alapján dolgozhatunk. A leíró állományban paraméterek is megadhatók, és ezek a parancssorba írt paramétereket felülbírálják.

A leíró állományban az új sorba írt információkat a Return billentyűvel kell elválasztani egymástól, mivel a szóköz és a TAB karaktereket figyelmen kívül hagyja. A # karakter a leíró állományban magyarázat elhelyezésére nyújt lehetőséget. Maga a leíró állomány tartalmazhat beágyazott leíró állományt, amelynek nevét a hívatkozás során a @ karakterrel kell kezdeni.

A leíró állományba írt utasítások végrehajtása idején azok megjelennek a képernyőn, ami a SILENT paraméterrel tíltható le.

b)POSITIONAL csatolóval

RTLINK [<tárgymodul lista>,
[<tölthető modul neve>],
[<memóriatérkép-állomány neve>],
[[<könyvtárlista>]
[<paraméterlista>]] [;]

Mint látható, az egyes állományneveket — az előzőektől eltérően — vesszővel kell elválasztani egymástól.

<tárgymodul lista> - megegyezik a FREEFORMAT csatolónál leírtakkal, azzal a különbséggel, hogy az egyes állományneveket szóközzel kell elválasztani egymástól, a vesszőt csak az utolsó név után kell kitenni.

<tölthető modul neve> - megegyezik a FREEFORMAT csatolónál leírtakkal.

<memóriatérkép-állomány neve> - egy memóriatérkép (MAP) állományt hoz létre, ami egy ASCII szövegállomány, és tartalmazza a szerkesztés lefolyásával kapcsolatos információkat. Csak akkor jön létre, ha a /MAP paramétert megadjuk.

<könyvtárlista> - megegyezik a FREEFORMAT csatolónál leírtakkal, azzal a különbséggel, hogy az egyes állományneveket szóközzel kell elválasztani egymástól.

<paraméterlista> - az egyes paramétereket a per (/) jellel kell egymástól elválasztani, és ezt közvetlenül kell a paraméter kulcsszavának követni.

Leíró állomány alkalmazása esetén az indítás formája:

RTLINK @<leíró állomány> <paraméterlista>

A leíró állománnyal kapcsolatos tudnivalók megegyeznek a FREEFORMAT csatolónál leírtakkal.

Figyelem! A POSITIONAL csatolónál a szerkesztőprogramot indító parancs kötött szintaktikája miatt a leíró állomány használatánál vigyázni kell! A leíró állomány végét a pontosvessző (;) jelzi. Folytatássorok szükségessége esetén a folytatandó sort a plusz (+) jellel kell lezárni.

A környezeti változó

Mint a fordítóprogramnál, a szerkesztőnél is megadhatók a gyakran alkalmazott paraméterek a környezeti változóban. Erre a célra a SET RTLINKCMD szolgál, amit célszerűen az AUTOEXEC.BAT-ban helyezünk el. Formája:

SET RTLINKCMD=<paraméterlista>

<paraméterlista> - a gyakran használt paraméterek listája, amelyek a szerkesztőprogram parancssorában megadott paraméterekkel felülbírálhatók. A FREEFORMAT csatoló esetén itt megadhatók a FILE és a LIBRARY információk is.

Példa:

Az állandóan használt könyvtár a SUELIB.LIB, a paraméterek közül a SILENT:

```
SET RTLINKCMD=LIB SUELIB.LIB SILENT
```

POSITIONAL csatoló esetén csak a paraméter adható meg:

```
SET RTLINKCMD=/POSITIONAL /SILENT
```

Az RTLINKCMD környezeti változó mellett az RTLINK.CFG konfigurációs állomány is tartalmazhatja a leggyakoribb paramétereket. A szerkesztőprogram elindulása után először az RTLINK.CFG állományt keresi az aktuális könyvtárban, majd abban a könyvtárban, ahol az RTLINK.EXE található. Ha talál ilyet, akkor a tartalmát figyelembe veszi. Ezt követően nézi az RTLINKCMD környezeti változó értékét, majd a parancssorban, és végül a leíró állományban megadott értékeket. Ez a sorrend egyben a paraméterek felülbírálhatósági sorrendjét is jelenti.

A visszatérési kód

Hiba, vagy felhasználó általi megszakítás (Ctrl-C vagy Ctrl-Break) esetén a DOS visszatérési kódja 1, illetve ugyancsak 1, ha a szerkesztőprogram lekezeletlen hívatkozással találkozik. Ezekben az esetekben a képernyőn hibajelzés jelenik meg, és a tölthető modul (.EXE) nem áll elő. Normális szerkesztés esetén a visszatérési kód 0.

A szerkesztőprogram által létrehozott állományok

A szerkesztőprogram a következő állományokat állítja elő:

- előszerkesztési könyvtár (.pll);
- előszerkesztés átalakító könyvtár (.plt);
- tölthető modul (.exe);
- memóriatérkép állomány (.map);
- overlay állományok (.ovl);
- ideiglenes állományok (.tmp);
- információs állományok (.inf).

Az *előszerkesztési könyvtár* a PRELINK paraméter megadása esetén jön létre. Kiterjesztése .PLL, ami nem írható át.

Az előszerkesztési könyvtár lényegében egy memóriába tölthető modul, amely a PLL paraméterrel megadott szerkesztés során a tárgymodulokból létrejövő .EXE állományhoz kapcsolódik. Tartalma a program futása során ugyanúgy viselkedik, mintha az .EXE része lenne. A rendszer az előszerkesztési könyvtárat sorrendben a következő helyeken keresi: abban a könyvtárban, ahol az .EXE található; a PLL környezeti változóval megadott könyvtárban; a LIB környezeti változóval megadott könyvtárban. Ha egyik helyen sem találja, hibajelzést ad.

Az előszerkesztési könyvtár jelentősen meggyorsíthatja programunk futását, és alkalmazása esetén csökken az .EXE állomány mérete, mivel a .LIB könyvtárak beszerkesztésével ellentétben a .PLL-ben található modulok nem képezik részét. Célszerű azokat a leggyakrabban hasz-

nált rutinokat ilyen előszerkesztési könyvtárba helyezni, amelyeket több programban is ugyanúgy használunk. Ez jelentős lemezhely-megtakarítást eredményez.

Az előszerkesztési könyvtárba bekerülő rutinokat a PRELINK paraméterrel kell szerkeszteni; az ezeket felhasználó programrendszerek szerkesztésénél pedig a PLL paraméterrel kell hivatkozni rá.

Az előszerkesztés átalakító könyvtár a .PLL állománnyal egyidejűleg jön létre, neve megegyezik azzal. Kiterjesztése .PLT, ami nem írható át. Ugyanabban a könyvtárban kell elhelyezkednie, ahol a .PLL található. A .PLT állomány az előszerkesztési könyvtár címeivel és szegmenseivel kapcsolatos információkat tartalmazza.

A tölthető modul az az állomány, amelynek memóriába töltésével a program futtatható. Hibátlan szerkesztés esetén jön létre, kiterjesztése .EXE, ami nem írható át. Indítás esetén keresése az alábbi helyeken és sorrendben történik: az aktuális könyvtárban, majd a SET PATH környezeti változóval leírt könyvtárakban. Ha nem található sehol, hibajelzés jelenik meg.

A memóriatérkép-állomány a MAP paraméter hatására jön létre. Kiterjesztésének alapértelmezése .MAP. Ez egy normál ASCII szövegállomány, amely a memóriahasználatról nyújt információkat.

Az overlay állományok a DYNAMIC vagy a SECTION paraméterek hatására jönnek létre. Alapértelmezés szerinti kiterjesztésük .OVL. Ugyanabban a könyvtárban kell lennie, ahol az .EXE állomány található.

Ideiglenes állomány a szerkesztés folyamán keletkezik, a későbbiekben erre nincs szükség. Ezért célszerű azokat egy könyvtárba átirányítani, és a későbbiek folyamán törölni. Az átirányítás formája:

```
SET TMP=<meghajtó[\könyvtárnév]>
```

Az átirányítást különösen akkor célszerű alkalmazni, ha kevés helyünk van a lemezen. Ekkor megtehetjük azt is, hogy az ideiglenes állományokat a memóriában kialakított virtuális lemezre írjuk, például:

```
SET TMP=D:
```

Az információs állomány az INCREMENTAL paraméter hatására jelentkezik abban az esetben, ha egy összehasonlító (inkrementáló) szerkesztést végzünk. Ennek ismertetése a paraméternél található. Az állományban arról jelennek meg információk, hogy az előző szerkesztés óta milyen változások történtek. Neve megegyezik az .EXE állományéval, kiterjesztése .INF.

A szerkesztőprogram paraméterei

A paraméterek — mint arról már korábban is szó volt — négy helyen adhatók meg, ezek (felülbírálásuk sorrendjében): az RTLINK.CFG konfigurációs állomány; az RTLINKCMD környezeti változó; az RTLINK parancssora; a leíró állomány. Ez a felülbírálási sorrend különösen az egymással ellentétes paraméterek megadása esetén érdekes. Például ha a leíró állomány a /NOIGNORECASE paramétert tartalmazza, míg a szerkesztő indítása a

```
RTLINK @LNKFILE /IGNORECASE
```

paranccsal történik, akkor a NOIGNORECASE paraméter lesz az érvényes.

A paraméterek neve rövidíthető. Az előző példa szerint érvényes az IG és a NOIG megadás is.

Azoknál a paramétereknél, ahol számot kell megadni, figyelni kell arra, hogy a FREEFORMAT csatoló esetén hexadecimális (16-os), míg POSITIONAL esetén tízes (10-es) számrendszer az alapértelmezés. A megadott szám más számrendszerben is lehet, ezt azonban jelölni kell. A szerkesztő az alábbi számrendszer jelöléseket ismeri és fogadja el:

Azonosító	Alapszám
.	Decimális (10)
B	Bináris (2)
H	Hexadecimális (16)
O	Oktális (8)

A számrendszer azonosítót a szám után közvetlenül kell írni, például: 11H - hexadecimális 11; 11. - decimális 11; 11B - bináris 11 stb. Számrendszer azonosítónak az utolsó karaktert értelmezi, például a 11BH hexadecimális 11B értéket jelent.

A FREEFORMAT csatolónál értelmezve van a NIL érték. Például a ("") üres érték helyettesíthető a NIL kulcsszóval.

A paraméterek megegyeznek a FREEFORMAT és a POSITIONAL csatolónál. Eltérés megadásuk módjában van: a POSITIONAL csatolónál a per (/) jelet kell elé tenni, míg a FREEFORMAT-nál a per jelet el kell hagyni. Az ismertetésnél a / jelet használjuk, ahol eltérés van a két csatoló szerinti megadás formájában, ott mind a két formát feltüntetjük. Az alapértelmezett értéket aláhúzással jelöljük.

Batch paraméterek

/BATCH | /NOBATCH

Ha a szerkesztőprogram nem talál egy .obj, .lib, vagy egy .plt állományt, akkor alapértelmezés szerint egy prompt után bekéri az elérési útvonalat. Rövidítése: NOBAT.

Ennek a funkciónak a letiltása a BATCH (rövidítve BAT). Ha ez a paraméter van érvényben, és valamelyik fent felsorolt kiterjesztésű állomány nem található, akkor hibajelzés jelentkezik.

Case-érzékenységi paraméterek

/IGNORECASE | /NOIGNORECASE

Alapértelmezés szerint nem tesz különbséget a kis- és nagybetűk között. Eszerint például a Suelib és a SUELIB nevek ugyanazt az állományt azonosítják. Rövidítése: IG.

A NOIGNORECASE (rövidítése NOIG) megadása esetén a fenti példában megadott nevek alapján a szerkesztőprogram két különböző állományt keres. Megadása nem javasolt, mivel sem a DOS, sem a Clipper, a C, illetve az Assembler nyelv nem tesz különbséget a kis- és nagybetűk között.

Konfigurációs paraméterek

/FREEFORMAT | /POSITIONAL

A már ismertetett felhasználói csatoló beállítása. Rövidítése FREE, illetve POSI.

A dinamikus overlay paraméterei

/DYNAMIC[:<overlay állomány>]
DYNAMIC [INTO <overlay állomány>]

A DYNAMIC paraméter a RESIDENT párja, a kettő közül ez az alapértelmezés. Rövidítése: DYN.

Hatására az ezt követően felsorolt .lib vagy .obj állományok egy .OVL kiterjesztésű dinamikus overlay ágba kerülnek. Ha nem adunk meg <overlay állomány>-t, akkor a modulok az .EXE részei lesznek. A C és Assembler nyelven írt rutinok dinamikus overlay ágba helyezését a Clipper 5.0 nem támogatja, ezért ilyen esetben a RESIDENT paramétert használjuk!

/RESIDENT

Rövidítése RES. A program vagy az előszerkesztési könyvtár tölthető moduljában egy belső overlay ágat hoz létre.

A különböző overlay technika együttesen is alkalmazhatók. Például a Pr1, Pr2 modulok dinamikus overlay-t képezve az .EXE állomány részei lesznek, a Pr3 az aktuális program belső overlay-je, míg a Pr4 az Ovlall külső overlay állomány része lesz (POSITIONAL csatoló esetén):

```
RTLINK /DYNAMIC Pr1 Pr2 /RESIDENT Pr3 /DYNAMIC:Ovlall Pr4;
```

Ugyanez a példa FREEFORMAT csatolónál:

```
RTLINK @Sajprog
```

ahol a Sajprog leíró állomány tartalma:

```
DYNAMIC
FILE Pr1, Pr2
RESIDENT
FILE Pr3
DYNAMIC INTO Ovlall
FILE Pr4
```

Információs és nyomkövetési paraméterek

/DEBUG

Nyomkövetési lehetőséget biztosít azáltal, hogy az éppen betöltött overlay ág nevét kiírja a képernyőre. Rövidítése DEB.

/HELP

A szerkesztőprogram paraméterezési lehetőségeit írja a képernyőre. Rövidítése H.

/MAP[:<memóriatérkép állomány>]
MAP [= <memóriatérkép állomány>] [map-paraméterlista>]

A szerkesztés végrehajtásáról, illetve az azzal kapcsolatos információkról ad listát. Rövidítése MA.

<memóriatérkép állomány> - annak az állománynak a neve, amelybe az információk beírásra kerülnek. A POSITIONAL csatolónál a /MAP a parancssor része, míg a FREEFORMAT-nál

paraméterként adható meg. Az állomány kiterjesztésének alapértelmezése, ha nem adjuk meg: .MAP.

<map-paraméterlista> - az határozható meg vele, hogy milyen összetételű memóriatérkép készüljön. Ha nem adunk meg semmit, az összes létező forma elkészül. A megadott paraméterek kombinálhatók egymással.

Paraméter	Jelentés
S	a szegmensek felsorolása kezdőcím szerint
N	a *public* változók neve szerinti lista kezdőcímükkel
A	a *public* változók listája kezdőcímük sorrendjében

Ha nem adjuk meg a MAP állomány nevét, az az elsőként feltüntetett tárgymodul nevét (példánkban Prog1.map) veszi fel (POSITIONAL csatolónál):

```
RTLINK Prog1 Prog2 /MAP:SN;
```

Ugyanez, ha a MAP állománynak Mapall.map nevet definiáljuk:

```
RTLINK Prog1 Prog2,, Mapall /MAP:SN;
```

A fenti példa FREEFORMAT csatoló esetén:

```
RTLINK FILE Prog1 Prog2 MAP SN
```

illetve

```
RTLINK FILE Prog1 Prog2 MAP=Mapall SN
```

/SILENT

Megadásával letilthatjuk a leíró állomány egyes utasításainak végrehajtásának megjelenítését a képernyőn. Rövidítése SI.

/VERBOSE[:<szint>]

Megadása esetén a képernyőn nyomon követhetjük a szerkesztés menetét: az éppen szerkesztés alatt álló modul neve, illetve a szerkesztés sorrendje jelenik meg. Összehasonlító szerkesztés esetén feltünteti, hogy az .EXE állomány hány százalékánál tart. Rövidítése VE.

<szint> - értéke 0, 1 vagy 2 lehet. A megjelenő információk tartalmát szabályozza.

Az összehasonlító szerkesztés paraméterei

/INCREMENTAL[:<arány>] | /NOINCREMENTAL

Alapértelmezés szerint a teljes programrendszert újraszerkeszti. Rövidítése NOIN.

Az INCREMENTAL (rövidítése INC) paraméter megadása esetén egy korábbi szerkesztési menet eredményével hasonlítja össze a jelenlegi állapotot. Csak azokat a modulok szerkeszti újra, amelyek változtak. Ez a szerkesztési idő rendkívüli mértékű csökkentését eredményezheti. A paraméterként megadható <arány> azt jelzi, hogy a változás milyen mértékénél kérjük a teljes újraszerkesztést. Ha nem adunk meg semmit, ennek alapértelmezése 25%.

Elérési paraméterek

/DEFAULTLIBRARYSEARCH | /NODEFAULTLIBRARYSEARCH

Az alapértelmezés szerinti érték (rövidítése DEFA) alapján a CLIPPER.LIB, EXTEND.LIB és a DBFNTX.LIB könyvtárak automatikusan beszerkesztésre kerülnek.

A NODEFAULTLIBRARYSEARCH (rövidítése NOD) esetén egyetlen könyvtár sem lesz automatikusan láncolva, arról nekünk kell gondoskodni.

/EXTDICTIONARY | /NOEXTDICTIONARY

Az alapértelmezés szerinti értéknél (rövidítése EX) minden azonos nevű *public* változóhivatkozást megvizsgál.

A NOEXTDICTIONARY (rövidítése NOE) nem terjeszti ki a változóhivatkozások keresését, ezzel elérhető, hogy például a programban átdefiniáljuk a könyvtárrutinban már alkalmazott *public* változót.

/STACK:<méret>

A STACK (rövidítése ST) paraméter segítségével felülírható a tárgymodulban specifikált veremtár mérete. A <méret> decimális pozitív egész szám, maximális értéke 65535 lehet.

A STACK paraméter egyaránt alkalmazható az előszerkesztés és a szerkesztés során.

Az előszerkesztés paraméterei

/EXCLUDE:<rutinnév>
EXCLUDE <rutinnév lista>

Az EXCLUDE (rövidítése EXC) paraméter segítségével az előszerkesztésre kerülő könyvtárból a megadott rutinok kihagyhatók. Ennek akkor van jelentősége, amikor ugyanaz a rutin be lett szerkesztve az .EXE állományba is. Ebben az esetben a két helyen szereplő rutin hibát eredményez. Ezért az előszerkesztési (.PLL) könyvtár újraszerkesztése során a konfliktust okozó rutin(oka)t az EXCLUDE paraméterrel ki kell venni. (Az előszerkesztés későbbi ismételt végrehajtása után az itt kiemelt rutinok ismét szerepelnek benne.)

A FREEFORMAT csatoló esetén egymás után vesszővel elválasztva több rutinnév is megadható, a POSITIONAL csatoló esetén minden egyes kiemelendő rutinhoz külön /EXCLUDE paramétert kell megadni.

/PLL:<előszerkesztési könyvtár>

A PLL paraméter segítségével tudjuk a korábban már létrehozott előszerkesztési (.PLL) könyvtár létezését a szerkesztő program tudomására hozni. Rövidítése PL. Egy előszerkesztési könyvtár hivatkozhat egy másikra, de csak egy szintig, azaz egy B.PLL könyvtár valamely rutinjához kapcsolódhat egy rutin az A.PLL előszerkesztési könyvtárból, de már egy C.PLL könyvtár nem hivatkozhat a B.PLL rutinjaira.

Az .EXE állomány létrehozásához a PLL paraméter hatására létrejön egy .PLT kiterjesztésű állomány is.

Fontos! Az előszerkesztésben résztvevő állományok (.PLL és .PLT) kiterjesztései nem változtathatók meg!

/PRELINK

Előszerkesztés indítását meghatározó paraméter. Rövidítése: PR. Alkalmazása esetén a szerkesztés után létrejövő állomány előszerkesztési könyvtár lesz. Az előszerkesztési könyvtárral kapcsolatosan korábban részletes írtunk.

/REFER:<rutinnév>
REFER <rutinnév lista>

A PRELINK paraméterrel korábban létrehozott előszerkesztési könyvtárba a későbbiek során új rutinok vihetők fel. Ezt biztosítja a REFER paraméter. A REFER kulcsszó utáni felsorolás tartalmazza az előszerkesztési könyvtárba bekerülő rutinok neveit. Rövidítése REF. A FREEFORMAT csatoló esetén egymás után több rutin neve adható meg, a POSITIONAL csatolónál annyi /REFER paramétert kell megadni, ahány rutint a könyvtárba fel kívánunk vinni.

A statikus overlay paraméterei

Figyelem! Ezek a paraméterek csak a FREEFORMAT csatoló esetén alkalmazhatók!

BEGINAREA

.
. **<utasítások>**
.

ENDAREA

A BEGINAREA kulcsszó jelzi egy statikus overlay terület kezdetét, amelynek leírása az ENDAREA kulcsszóig tart. A két kulcsszó között szereplő összes ág (egy ágat egy SECTION paraméter határoz meg) ugyanahhoz az overlay területhez fog tartozni. Lehetőség van egynél több, illetve egymásba ágyazott overlay területek létrehozására. Egymásba ágyazás esetén be kell tartani a struktúrakialakítás szabályait: a BEGINAREA és ENDAREA kulcsszavakat páronként kell alkalmazni. A kulcsszavak rövidítései: BE és EN.

MODULE <modulnév lista>

A MODULE paraméter a megadott modulokat áthelyezi az aktuális statikus overlay ágba. Rövidítése MO. Ez a paraméter felülbírálja a FILE és a LIBRARY paraméterekben megadottakat.

A <modulnév lista>-ban szereplő modulneveket vesszővel kell egymástól elválasztani. Ha üres nevet akarunk megadni, használjuk a NIL kulcsszót.

PRELOAD

A PRELOAD paraméter hatására az aktuális overlay ág még a program indulása előtt betöltődik a memóriába. (Normál esetben csak a főprogram rezidens része töltődik be az indulás előtt.) Rövidítése PRE.

Példa a PRELOAD paraméter használatára:

```
SECTION = Overag1
PRELOAD
FILE Prog1
SECTION Overag2 INTO Kulsoag
PRELOAD
FILE Prog2
```

STATIC [= <overlay ág neve>] [INTO <overlay állománynév>]

A SECTION paraméter hatására jön létre egy statikus overlay ág. Ezen belül nemcsak Clipper tárgymodulok nevei adhatók meg (ellentétben a dinamikus overlay-vel). Rövidítése SEC.

Ha egy Clipper modult adunk meg egy statikus belső overlay ágon, az ugyanúgy viselkedik, mintha dinamikus overlay-t alkalmaztunk volna.

Külső overlay esetén a SECTION kulcsszó sorában meg kell adni az INTO kulcsszót és a külső overlay állomány nevét is, amelynek .OVL kiterjesztéssel kell rendelkezni. Ha az INTO kulcsszó nem szerepel, belső overlay jön létre.

Figyelem! A SECTION és az INTO egymáshoz tartozó kulcsszavak, amelyek közé más paraméter nem kerülhet! Az overlay ág definiálása (SECTION ... INTO) után következhet a FILE és minden más paraméter.

Az előszerkesztés

Mivel az előszerkesztés a Clipper 5.0 teljesen új része, érdemes az eddig ismertetettek mellett bővebben is foglalkozni vele.

Az előszerkesztés lényegében egy szerkesztési menet, amely a PRELINK paraméter hatására nem .EXE állományt, hanem .PLL kiterjesztésű előszerkesztési könyvtárt hoz létre. Ez a könyvtár egy tölthető modul, amely azonban csak egy rá hivatkozó .EXE állománnyal együtt hajtható végre.

Az előszerkesztést követi egy szerkesztés, amelyben a korábban már beszerkesztett .PLL állományra a PLL paraméterrel kell hivatkozni.

Eldöntendő kérdés, hogy milyen modulok kerüljenek az előszerkesztési könyvtárba? Ennek meghatározását az befolyásolja, hogy a szerkesztési időt, vagy a lefoglalt lemezterület nagyságát kívánjuk csökkenteni. Ha a szerkesztési idő csökkentése a cél, akkor az összes stabil, már nem változó modult (beleértve a CLIPPER.LIB és EXTEND.LIB szükséges részeit is) helyezzük az előszerkesztési könyvtárba, így az ismételt szerkesztések során jelentős időmegtakarítás érhető el, mivel az előszerkesztési könyvtár részei nem vesznek részt a szerkesztési fázisban. Ennek a megoldásnak az a valószínű hátránya, hogy minden .EXE állományhoz külön (ilyenformán elég nagy) előszerkesztési könyvtárat kell létrehozni.

Ha a lemezterület csökkentése a cél, akkor a minden .EXE állományban, vagy azok nagy többségében használatos modulokat helyezzük egy közös .PLL könyvtárba. A különböző programrendszerek szerkesztésekor ezek a modulok nem lesznek az egyes .EXE állományok részei (csak egyszer szerepelnek a lemezen).

Az előszerkesztési könyvtár létrehozásának menete

Ha eldöntöttük, hogy előszerkesztési könyvtárat hozunk létre, akkor a következő lépéseket hajtsuk végre:

- határozzuk meg a programrendszert alkotó tárgymodulokat;
- határozzuk meg a használni kívánt könyvtárakat;
- határozzuk meg a tárgymodulok külső hivatkozásait;
- határozzuk meg azokat a modulokat, amelyeket betölthetővé kívánunk tenni;

- tervezzük meg az előszerkesztési könyvtár tartalmát;
- határozzuk meg, mely rutinokat helyezzük az előszerkesztési könyvtárba;
- hozzuk létre az előszerkesztési könyvtárat.

Az előszerkesztési könyvtár megadásának formája (POSITIONAL csatoló esetén):

RTLINK [<tárgymodul lista>,
[<létrejövő állomány neve>],
[<memóriatérkép állomány neve>],
[<könyvtárlista>]
/PRELINK
[/REFER: <rutinnév> [/REFER:<rutinnév>...]]
[/EXCLUDE:<rutinnév> [/EXCLUDE:<rutinnév>...]]];

FREEFORMAT csatoló esetén:

RTLINK [FILE <tárgymodul lista>
[OUTPUT <létrejövő állomány neve>]
[LIBRARY <könyvtárlista>]
PRELINK
[REFER <rutinnevek>]
[EXCLUDE <rutinnevek>]

A fenti példa szerint a <tárgymodul lista>-ban felsorolt modulok lesznek az előszerkesztési könyvtár részei, neve a <létrejövő állomány neve> lesz, ha megadtuk. Ha nem adtuk meg, akkor a tárgymodul listában első helyen szereplő modul neve. Kiterjesztése .PLL. Erre a névre kell hivatkozni a normál szerkesztési menet PLL paraméterében.

A <tárgymodul lista>, ami lehet egy könyvtár (beleértve a CLIPPER.LIB-et és az EXTEND.LIB-et is) tartalmazhat rutingyűjteményt, amelyből csak meghatározott nevűekre van szükségünk. Ezeknek a nevét kell a REFER paraméter után felsorolni. Ha a rutingyűjtemény nagyobb részét használjuk, akkor pedig azoknak a neveit soroljuk fel az EXCLUDE paraméter után, amelyekre nincs szükségünk.

(Annak meghatározására, hogy a CLIPPER.LIB és az EXTEND.LIB könyvtárakból milyen rutinokat hív meg a programunk ajánlhatjuk a következő módszert: Hajtsunk végre egy normál szerkesztést úgy, hogy nem adunk meg könyvtárnevet, és a NODEFAULTLIBRARYSEARCH paramétert alkalmazzuk. A tárgymodulok lekezeletlen külső hivatkozásai — amelyeket egyébként a könyvtárakban keres — üzenetként megjelennek. A megjelenítés érdekében alkalmazzuk a VERBOSE paramétert. Ha a szerkesztési idő rövidítése a célunk, akkor az így kiszűrt rutinokat helyezzük el az előszerkesztési könyvtárban.)

Szabvány osztályok

A Clipper 5.0 egy lépést tett az objektumorientált programozás irányába. Hangsúlyozattan még nem objektumorientált programozási nyelv, de annak elemeit már alkalmazza. A nyelvben megjelent egy új adattípus — az *objektum* — fogalma. Saját objektumosztályokat még nem hozhatunk létre, de a négy szabvány osztály objektumaival végezhető műveletek elég széleskörűek. Mivel teljesen új alkalmazási lehetőségről van szó, az érdeklődőknek feltétlenül ajánlom a SAMPLE, illetve a SYS alkönyvtárban található forrásnyelvű példaprogramok megtekintését. A szabvány osztályok kezelésének megismerése mellett sok egyéb programozási fogás is elleshető ezekből. A szabvány osztályok utasításkészletének részletes ismertetését terjedelmi okokból nem közöljük, csupán a funkcióikra hívjuk fel a figyelmet. Ezek részletes leírását a programnyelvhez mellékelt *Clipper 5.0 Reference* dokumentáció 6. fejezete tartalmazza.

A Clipper 5.0 szabvány osztályai:

- hibakezelő osztály (Error Class);
- adatbeviteli osztály (GET Class);
- adatáttekintési osztály (TBrowse Class);
- adatmegjelenítési osztály (TBColumn Class).

Hibakezelő osztály

Error Class

Egy hibaobjektum a jelentkező hibajelzésről ad információkat. Amikor egy hibajelzés jelentkezik, létrejön egy hibaobjektum, és ez mint egy argumentumérték átadódik az ERRORBLOCK() függvénynek. A hibaobjektum a BEGIN SEQUENCE utasítás BREAK kulcsszavának alkalmazásával egy RECOVER utasításhoz irányítható, ahol lokális hibakezelésre használható fel.

Külső hívatkozási osztályok

args	Függvénytömb vagy operátorargumentumok
canDefault	Alapértelmezés visszaállítását jelzi
canRetry	Hiba utáni helyes visszaállítást jelzi
canSubstitute	Hiba után új eredmény beállítását jelzi
cargo	Felhasználó által definiált változó
description	A hibakód karakteres leírása
filename	A hibával összefüggő állomány neve
genCode	A Clipper-hiba kódja
operation	A hibát eredményező művelet leírása
osCode	Az operációs rendszerhiba kódja
subCode	Alrendszer-specifikus hibakód
subSystem	Az alrendszerben generált hiba leírása
tries	A hibás művelet javítási kísérleteinek száma

Példák:

Egy USE parancs során fellépő hiba lekezelése:

```
#include "Error.ch"
#command RETRY   => RETURN (.T.)
#command RESUME  => RETURN (.F.)
//
FUNCTION Hibakez( hibaobj )
   //
   IF hibaobj:genCode == EB_OPEN .AND.;
      hibaobj:canDefault .AND.;
      NETERR()
      //
      RESUME
   ENDIF
.
. <utasítások>
.
RETURN NIL
```

Annak illusztrálása, hogyan kell a hibakezelő egy hibaobjektumát a RECOVER utasításnak átadni a további feldolgozás érdekében:

```
LOCAL objlocal, uthandler
//
uthandler := ERRORBLOCK({ |objErr| sajhand(objErr, .T.) })
BEGIN SEQUENCE
.
. <utasítások>
.
RECOVER USING objlocal
.
. <üzenet küldése a hibahelynek és hibakezelés>
.
END
//
ERRORBLOCK(uthandler)


FUNCTION sajhand( objError, helyihand )
//
IF helyihand
   BREAK objError
ENDIF
.
. <hibakezelö utasítások>
.
RETURN NIL
```

Lásd még: A hibakódok könnyebb kezelésére szolgál az Error.ch állomány. Az alapértelmezés szerint hibakezelő (handler) forrásnyelvű leírását az Errorsys.prg tartalmazza. A hi-

bakezelő osztállyal, illetve a hibakezeléssel kapcsolatos információk találhatók még a BEGIN SEQUENCE, DOSERROR(), ERRORBLOCK() és NETERR() ismertetésénél.

Adatbeviteli osztály

GET Class

A GET objektum az adatok beviteli és szerkesztési mechanizmusának lelke. Szorosan kapcsolódik a @...GET és READ parancsokhoz. A GET objektumok segítségével történik az adatok formázása és szerkesztése, beleértve a kurzormozgatást és a helyességellenőrzést is. A helyességellenőrzés a felhasználó által megadott kódblokkok, míg a kijelzés formázása a szabvány PICTURE kódkészlettel valósul meg.

A GET objektum az egyes változókhoz kapcsolódik, de közvetlenül nem tudja manipulálni azokat, csak egy kódblokkon keresztül. Amikor egy szabvány @...GET paranccsal egy GET objektumot létrehozunk, automatikusan létrejön egy kódblokk is, amely lehetővé teszi a változó manipulálását a megadott parancsok szerint. A GET objektumot a GetNew() függvényben megadott kódblokkal lehet kezelni.

A GET objektum átveszi a változó értékét tartalmazó kódblokkot. Ezt az értéket megjeleníthető formára konvertálja, és bemásolja a szerkesztési pufferbe. A szerkesztési puffer tartalma jelenik meg a képernyőn, amely már szerkeszthető. Az itt végrehajtott módosítások érvényre jutnak a GET változó értékében.

Külső hivatkozási változók

badDate	Jelzi a hibás dátumot a szerkesztési pufferben
block	A GET-et a változóval összekötő kódblokk
buffer	A szerkesztési puffert meghatározó karakteres érték
cargo	Felhasználó által definiált változó
changed	Jelzi a puffermódosítást
col	A Get oszlopszáma
colorSpec	A színattributum jelzése
decPos	A tizedespont helye a szerkesztési pufferben
hasFocus	A GET-objektum input fókuszának jelzése
name	A Get változó neve
original	A GET eredeti értékét tartalmazó sztring
picture	PICTURE sztring
pos	A szerkesztési puffer kurzorpozíciója
postBlock	Az új értéket ellenőrző kódblokk
preBlock	A szerkesztést meghatározó kódblokk
rejected	Jelez, ha utoljára karakterbeszúrás/felülírás történt
row	A Get sorának száma
type	A Get változó adattípusa
typeOut	Jelzi, ha a kurzor kiment a szerkesztési pufferből

Külső eljárások

Állapotváltoztató eljárások

assign()	Átadja a szerkesztési puffer tartalmát a Get változónak
display()	A Get megjelenítése a képernyőn
killFocus()	Az input fókusz törlése a GET objektumból
reset()	Törli a GET belső állapotinformációit
setFocus()	Input fokuszt ad a GET objektumnak
undo()	A változó értékét visszaállítja a Get:original szerint
updateBuffer()	Módosítja a szerkesztési puffer tartalmát
varGet()	A GET változó aktuális értéke
varPut()	A GET értékét átadja a változónak

Kurzormozgató eljárások

end()	A kurzort a sor végére viszi
home()	A kurzort a sor elejére viszi
left()	A kurzort egy hellyel balra viszi
right()	A kurzort egy hellyel jobbra viszi
toDecPos()	A kurzort a tizedespont utánra állítja
wordLeft()	A kurzort egy szóval balra viszi
wordRight()	A kurzort egy szóval jobbra viszi

Szerkesztési eljárások

backspace()	A kurzortól balra álló karaktert törli
delLeft()	A kurzortól balra álló karaktert törli
delRight()	A kurzortól jobbra álló karaktert törli
delWordLeft()	A kurzortól balra álló szót törli
delWordRight()	A kurzortól jobbra álló szót törli

Szövegbeviteli eljárások

insert()	Karakter beszúrása a szerkesztési pufferbe
overStrike()	Karakter átírása a szerkesztési pufferben

Példa:

Létrehozunk egy GET objektumot, majd módosítjuk a READMODAL() függvénnyel:

```
LOCAL valt := SPACE(10)
//
// GET objektum létrehozása
objGet := GetNew()
//
objGet:row := 10
objGet:col := 10
//
// A változó nevének és a kódblokknak a definiálása
objGet:name := "valt"
objGet:block := { |ertek| IF (PCOUNT() > 0,;
   valt :=ertek, valt) })
```

```
objGet:picture := "@!"
objGet:colorSpec := "BG+/B, W+/BG"
objGet:postBlock := { |ertek| !EMPTY(ertek) }
// A GET objektum szerkesztése
READMODAL({objGet})
```

Lásd még: A forrásnyelvű leírást a Getsys.prg tartalmazza. Utalások: @...GET, READ, READMODAL(), READVAR(), SETCOLOR()

Az adatáttekintési osztály

TBrowse Class

A TBrowse objektum a táblázatos formában történő adatáttekintés és formázás hatékony és rugalmas eszköze. Egy Tbrowse objektum egy vagy több TBColumn objektummal áll kapcsolatban. Egy TBColumn objektum az adatáttekintő táblázat egy oszlopával kapcsolatos információkat hordozza.

Egy művelet hatására a TBrowse objektum átveszi a kódblokk értéke szerinti adatokat, amelyek sorokba és oszlopokba rendezve megjelennek a képernyő kijelölt területén. A TBrowse objektum egy belső, úgynevezett *browse kurzorral* rendelkezik. Az az adat, amelyre a browse kurzor mutat, alászíneződik, ugyanakkor az aktuális kurzor ennek az adatnak az első karakterén áll.

Osztályszintű függvények

TBrowseNew()	Egy új TBrowse objektum létrehozása
TBrowseDB()	Egy új TBrowse objektum létrehozása adatbázishoz

Külső hivatkozási változók

autoLite	Alászínezést ellenőrző logikai érték
cargo	Felhasználó által definiált változó
colCount	A browse-oszlop száma
colorSpec	Színtáblázat
colPos	Az aktuális kurzoroszlop
colSep	Az oszlopelválasztó karakter
freeze	A befagyasztott oszlopok száma
goBottomBlock	A TBrowse:goBottom() által létrehozott kódblokk
goTopBlock	A TBrowse:goTop() által létrehozott kódblokk
headSep	A fejrészt elkülönítő karakter
hitBottom	Az adatterület végét jelzi
nBottom	A browse-ablak alsó sora
nLeft	A browse-ablak baloldali oszlopa
nRight	A browse-ablak jobboldali oszlopa
nTop	A browse-ablak felső sora
rowCount	A browse-ablakban látható adatsorok száma
rowPos	Az aktuális kurzorpozíció sora
skipBlock	Az adatok áthelyezésére szolgáló kódblokk
stable	Állandó TBrowse objektum esetén jelez

Külső eljárások

Kurzormozgatások

down()	A kurzort egy sorral lejjebb viszi
end()	A kurzort a látható adat végére viszi
goBottom()	Az adatállomány végére megy
goTop()	Az adatállomány elejére megy
home()	A kurzort a látható adat elejére viszi
left()	A kurzort egy oszloppal balra viszi
pageDown()	Lapozás lefelé
pageUp()	Lapozás fölfelé
panEnd()	A kurzort az adatoszlop végére viszi
panHome()	A kurzort az adatoszlop elejére viszi.
panLeft()	Balra mozgás a kurzorpozíció változása nélkül
panRight()	Jobbra mozgás a kurzorpozíció változása nélkül
right()	A kurzort egy oszloppal jobbra viszi
up()	A kurzort egy sorral feljebb viszi

Vegyes eljárások

addColumn()	Egy TBColumn objektum hozzáadása a TBrowse objektumhoz
colorRect()	Módosítja a cellákra merőleges csoport színét
configure()	Átkonfigurálja a TBrowse objektum belső környezetét
deHilite()	Az aktuális cella fényességének kioltása
getColumn()	A megadott TBColumn objektum megadása
hilite()	Az aktuális cella fényesítése
refreshAll()	Az összes adat felfrissítése a következő stabilizáláskor
refreshCurrent()	Az aktuális sor felfrissítése a következő stabilizálásra
setColumn()	Egy TBColumn objektum felcserélése egy másikkal
stabilize()	Összehasonlító stabilizálás

Az adatáttekintési osztály alkalmazására szemléletes példa található a SOURCE\SAMPLE alkönyvtárban TbDemo.prg néven.

Lásd még: TBColumn osztály, BROWSE(), DBEDIT(), SETCOLOR()

Az adatmegjelenítési osztály

TBColumn Class

Egy TBColumn objektum valamely TBrowse objektum egy adatoszlopával kapcsolatos információkat tartalmazza.

Osztályszintű függvény

TBColumnNew() Egy új TBColumn objektum létrehozása

Külső hívatkozási változók

block	Az oszlop értékét tartalmazó kódblokk
cargo	Felhasználó által definiált változó
colorBlock	Az adatmezők színét meghatározó kódblokk
colSep	Az oszlopelválasztó karakter
defColor	A színtáblázat numerikus indexeinek tömbje
footing	Lábjegyzet oszlop
footSep	A lábjegyzetet elkülönítő karakter
heading	Fejrész oszlop
headSep	A fejrészt elkülönítő karakter
width	A képernyő oszlopszélessége

Példa:

Annak illusztrálására, hogyan hozzunk létre egy TBrowse objektumot, és azt hogyan töltsük fel a megfelelő TBColumn objektumokkal:

```
USE Cimek NEW
//
// A TBrowse objektum létrehozása
objBrowse := TBrowseDB(1, 1, 23, 79)
//
// TBColumn objektumok létrehozása és hozzáadása
objBrowse:addColumn(TBColumnNew( "Neve", {|| Cimek->Neve} ))
objBrowse:addColumn(TBColumnNew( "Cime", {|| Cimek->Cime} ))
objBrowse:addColumn(TBColumnNew( "Lakas", {|| Cimek->Lakcim} ))
.
. <utasítások>
.
CLOSE Cimek
```

A TBrowse objektum alkalmazására egy igen szemléletes példa található a SOURCE\SAMPLE könyvtárban TbDemo.prg néven.

Lásd még: TBrowse osztály, BROWSE(), DBEDIT()

Függelék (kódtáblázatok)

INKEY kódok

Billentyűk

Kurzormozgatás	Kód	Inkey.ch
fel nyíl, Ctrl-E	5	K_UP
le nyíl, Ctrl-X	24	K_DOWN
balra nyíl, Ctrl-S	19	K_LEFT
jobbra nyíl, Ctrl-D	4	K_RIGHT
Home, Ctrl-A	1	K_HOME
End, Ctrl-F	6	K_END
PgUp, Ctrl-R	18	K_PGUP
PgDn, Ctrl-C	3	K_PGDN
Ctrl-balra nyíl, Ctrl-Z	26	K_CTRL_LEFT
Ctrl-jobbra nyíl, Ctrl-B	2	K_CTRL_RIGHT
Ctrl-Home, Ctrl-]	29	K_CTRL_HOME
Ctrl-End, Ctrl-W	23	K_CTRL_END
Ctrl-PgUp, Ctrl-kötőjel	31	K_CTRL_PGUP
Ctrl-PgDn, Ctrl-^	30	K_CTRL_PGDN
Ctrl-Return	10	K_CTRL_RET
Ctrl-ScrollLock	3	K_CTRL_C
Esc, Ctrl-[	27	K_ESC
Return, Ctrl-M	13	K_RETURN
Enter, Ctrl-M	13	K_ENTER

Szerkesztés	Kód	Inkey.ch
Del, Ctrl-G	7	K_DEL
Tab, Ctrl-I	9	K_TAB
Shift-Tab	271	K_SH_TAB
Ins, Ctrl-V	22	K_INS
Backspace, Ctrl-H	8	K_BS
Ctrl-Backspace	127	K_CTRL_BS

Funkcióbillentyűk	Kód	Inkey.ch
F1, Ctrl-\	28	K_F1
F2	-1	K_F2
F3	-2	K_F3
F4	-3	K_F4
F5	-4	K_F5
F6	-5	K_F6
F7	-6	K_F7
F8	-7	K_F8
F9	-8	K_F9
F10	-9	K_F10

Alt és Ctrl billentyűk

Billentyű	Kód	Konstans	Billentyű	Kód	Konstans
Alt-A	286	K_ALT_A	Alt-N	305	K_ALT_N
Alt-B	304	K_ALT_B	Alt-O	280	K_ALT_O
Alt-C	302	K_ALT_C	Alt-P	281	K_ALT_P
Alt-D	288	K_ALT_D	Alt-Q	272	K_ALT_Q
Alt-E	274	K_ALT_E	Alt-R	275	K_ALT_R
Alt-F	289	K_ALT_F	Alt-S	287	K_ALT_S
Alt-G	290	K_ALT_G	Alt-T	276	K_ALT_T
Alt-H	291	K_ALT_H	Alt-U	278	K_ALT_U
Alt-I	279	K_ALT_I	Alt-V	303	K_ALT_V
Alt-J	292	K_ALT_J	Alt-W	273	K_ALT_W
Alt-K	293	K_ALT_K	Alt-X	301	K_ALT_X
Alt-L	294	K_ALT_L	Alt-Y	277	K_ALT_Y
Alt-M	306	K_ALT_M	Alt-Z	300	K_ALT_Z
Ctrl-A	1	K_CTRL_A	Ctrl-N	14	K_CTRL_N
Ctrl-B	2	K_CTRL_B	Ctrl-O	15	K_CTRL_O
Ctrl-C	3	K_CTRL_C	Ctrl-P	16	K_CTRL_P
Ctrl-D	4	K_CTRL_D	Ctrl-Q	17	K_CTRL_Q
Ctrl-E	5	K_CTRL_E	Ctrl-R	18	K_CTRL_R
Ctrl-F	6	K_CTRL_F	Ctrl-S	19	K_CTRL_S
Ctrl-G	7	K_CTRL_G	Ctrl-T	20	K_CTRL_T
Ctrl-H	8	K_CTRL_H	Ctrl-U	21	K_CTRL_U
Ctrl-I	9	K_CTRL_I	Ctrl-V	22	K_CTRL_V
Ctrl-J	10	K_CTRL_J	Ctrl-W	23	K_CTRL_W
Ctrl-K	11	K_CTRL_K	Ctrl-X	24	K_Ctrl_X
Ctrl-L	12	K_CTRL_L	Ctrl-Y	25	K_Ctrl_Y
Ctrl-M	13	K_CTRL_M	Ctrl-Z	26	K_CTRL_Z

Shift, Alt és Ctrl-funkcióbillentyűk

Billentyű	Kód	Konstans	Billentyű	Kód	Konstans
Shift-F1	-10	K_SH_F1	Ctrl-F1	-20	K_CTRL_F1
Shift-F2	-11	K_SH_F2	Ctrl-F2	-21	K_CTRL_F2
Shift-F3	-12	K_SH_F3	Ctrl-F3	-22	K_CTRL_F3
Shift-F4	-13	K_SH_F4	Ctrl-F4	-23	K_CTRL_F4
Shift-F5	-14	K_SH_F5	Ctrl-F5	-24	K_CTRL_F5
Shift-F6	-15	K_SH_F6	Ctrl-F6	-25	K_CTRL_F6
Shift-F7	-16	K_SH_F7	Ctrl-F7	-26	K_CTRL_F7
Shift-F8	-17	K_SH_F8	Ctrl-F8	-27	K_CTRL_F8
Shift-F9	-18	K_SH_F9	Ctrl-F9	-28	K_CTRL_F9
Shift-F10	-19	K_SH_F10	Ctrl-F10	-29	K_CTRL_F10
Alt-F1	-30	K_ALT_F1			
Alt-F2	-31	K_ALT_F2			
Alt-F3	-32	K_ALT_F3			
Alt-F4	-33	K_ALT_F4			
Alt-F5	-34	K_ALT_F5			
Alt-F6	-35	K_ALT_F6			
Alt-F7	-36	K_ALT_F7			
Alt-F8	-37	K_ALT_F8			
Alt-F9	-38	K_ALT_F9			
Alt-F10	-39	K_ALT_F10			

PICTURE kódok

SAY PICTURE funkciók

Kód	Jelentés
B	Számok balra igazítása
C	Pozitív számok után CR (credit) írása
D	A dátum kiírása a SET DATE formátumában
E	A dátum brit alakban
R	Maszktól eltérő karakterek beszúrása
X	Negatív számok után DB (debit) írása
Z	A vezető nullák szóközökkel helyettesítése
(	A negatív számok zárójelbe helyezése
!	Nagybetűre alakítás

SAY PICTURE maszkkarakterek

Kód	Jelentés	Kód	Jelentés
A	Tetszőleges adattípus	!	Nagybetűre alakítása
N	Tetszőleges adattípus	$	A vezető szóköz helyén a $ jel
X	Tetszőleges adattípus	*	A vezető szóköz helyén a *
9	Tetszőleges adattípus	.	A tizedes pont helye
#	Tetszőleges adattípus	,	A számok tagolására szolgáló jel
L	Logikai érték "T" vagy "F" alakban		
Y	Logikai érték "Y" vagy "N" alakban		

GET PICTURE funkciók

Jel	Típus	Jelentés
A	C	Csak alfabetikus karakterek
B	N	Számok balra igazítva
C	N	A pozitív számok után CR (credit) jelzés
D	D,N	A dátumok a SET DATE formátumában
E	D,N	Dátumnál a nap és a hónap felcserélve, függetlenül a SET DATE formától Numerikus adatnál a vesszők és a pontok felcserélve (megadni rendesen kell!)
K	All	Ha elsőként nem kurzormozgató billentyűt nyomunk le, törli a mező tartalmát
R	C	Maszktól eltérő karakterek beszúrása
S<n>	C	Vizszintes görgetés; <n> egész szám, az egysoros ablak hossza
X	N	A negatív számok után a DB (debit) jelzés
Z	N	A vezető nullák szóközökkel feltöltése
(	N	A negatív számok zárójelben megjelenítése vezető szóközökkel
)	N	A negatív számok zárójelben megjelenítése vezető szóközök nélkül
!	C	Nagybetűre alakítás

GET PICTURE maszkkarakterek

Kód	Jelentés
A	Csak alfabetikus karakter
N	Alfabetikus és numerikus karakter
X	Tetszőleges karakter
9	Csak számjegyek (tetszőleges adattípusban) előjelekkel együtt
#	Számjegyek, szóközök és előjelek tetszőleges adattípusban
L	Csak T, F, Y vagy N
Y	Csak Y vagy N
!	Nagybetűs alakra konvertálás
$	Numerikus adatnál a vezető szóközök $ jellel helyettesítése
*	Numerikus adatnál a vezető szóközök *-gal helyettesítése
.	A tizedes pont helye
,	A számok tagolására szolgáló jel

Foglalt szavak

Foglalt szavak és függvénynevek

IF	ABS	DATE	EXP
ELSE	ASC	DAY	FCOUNT
ELSEIF	AT	DELETED	FIELDNAME
ENDIF	BOF	DEVPOS	FILE
WHILE	CDOW	DOW	FLOCK
ENDCASE	CHR	DTOC	FOUND
ENDDO	CMONTH	DTOS	IF
FUNCTION	COL	EMPTY	IIF
PROCEDURE	CTOD	EOF	INKEY
INT	MONTH	ROW	TIME
LASTREC	PCOL	RTRIM	TRANSFORM
LEN	PCOUNT	SECONDS	TRIM
LOCK	PROW	SELECT	TYPE
LOG	RECCOUNT	SETPOS	UPPER
LOWER	RECNO	SPACE	VAL
LTRIM	REPLICATE	SQRT	VALTYPE
MAX	RLOCK	STR	WORD
MIN	ROUND	SUBSTR	YEAR

- A foglalt szavak nem használhatók változók, eljárások vagy felhasználói függvények elnevezéseként.
- A foglalt függvénynevek a fordítóprogram részét képezik, azokat nem szabad átdefiniálni.
- A foglalt szavak négy vagy több karaktert tartalmazó részei is foglaltnak számítanak.
- Az egy vagy két aláhúzásjellel (_) kezdődő azonosítónevek, mint rendszerazonosítók szintén foglaltnak tekintendők.

Környezetleíró paraméterek

Az AUTOEXEC.BAT-ban beállítható környezetleíró paraméterek

Parancs	Jelentés
SET CLIPPER	Futási (runtime) konfiguráció (runtime)
SET CLIPPERCMD	Fordítóprogram paraméterek alapértelmezései (compiler)
SET INCLUDE	Kapcsolódó (header) állományok keresési útvonala (compiler)
SET LIB	A (.LIB, .PLT, .PLL) könyvtárak keresési útvonala (linker, runtime)
SET OBJ	A tárgyállományok (.OBJ) keresési útvonala (linker)
SET PLL	A .PLT és .PLL állományok keresési útvonala (linker, runtime)
SET RMAKE	Az RMAKE parancssor paraméterek alapértelmezései (make)
SET RTLINKCMD	Az .RTLink parancssor paraméterek alapértelmezései (linker)
SET TMP	Az ideiglenes állományok elhelyezése (compiler, linker)

Tárgymutató

Előfordító direktívák

#command/#translate 59
#define 61
#ifdef 64
#ifndef 65
#include 66
#undef 67

Függvények

AADD() 171
ABS() 171
ACHOICE() 171
ACLONE() 173
ACOPY() 173
ADEL() 174
ADIR()* 174
AEVAL() 175
AFIELDS()* 175
AFILL() 176
AINS() 176
ALIAS() 177
ALLTRIM() 177
ALTD() 177
ARRAY() 178
ASC() 178
ASCAN() 178
ASIZE() 179
ASORT() 179
AT() 179
BIN2I() 180
BIN2L() 180
BIN2W() 180
BOF() 180
BREAK() 181
BROWSE() 181
CDOW() 181
CHR() 181
CMONTH() 182
COL() 182
CTOD() 182
CURDIR() 182
DATE() 183
DAY() 183
DBCREATE() 183
DBEDIT()* 184
DBEVAL() 185
DBF()* 186
DBFILTER() 186
DBRELATION() 186
DBRSELECT() 187
DBSTRUCT() 187
DELETED() 187
DESCEND() 188
DEVICEPOS() 188
DEVOUT() 188
DIRECTORY() 188
DISKSPACE() 189
DISPOUT() 189
DOSERROR() 190
DOW() 190
DTOC() 190
DTOS() 190
EMPTY() 191
EOF() 191
ERRORBLOCK() 191
ERRORLEVEL() 192
EVAL() 192
EXP() 192
FCLOSE() 192
FCOUNT() 193
FCREATE() 193
FERASE() 193
FERROR() 194
FIELD() 194
FIELDBLOCK() 195
FIELDGET() 195
FIELDPUT() 195
FIELDWBLOCK() 195
FILE() 195
FKLABEL()* 196
FKMAX()* 196
FLOCK() 196
FOPEN() 196
FOUND() 197
FREAD() 197
FREADSTR() 198
FRENAME( ) 198
FSEEK() 198
FWRITE() 199
GETACTIVE() 199

GETENV() 199
HARDCR() 199
HEADER() 200
IF() 200
INDEXEXT() 200
INDEXKEY() 200
INDEXORD() 201
INKEY() 201
INT() 201
ISALPHA() 201
ISCOLOR() 202
ISDIGIT() 202
ISLOWER() 202
ISPRINTER() 202
ISUPPER() 203
I2BIN() 203
LASTKEY() 203
LASTREC() 203
LEFT() 204
LEN() 204
LOG() 204
LOWER() 205
LTRIM() 205
LUPDATE() 205
L2BIN() 205
MAX() 206
MAXCOL() 206
MAXROW() 206
MEMOEDIT() 206
MEMOLINE() 207
MEMOREAD() 208
MEMORY() 208
MEMOTRAN() 209
MEMOWRIT() 209
MEMVARBLOCK() 209
MIN() 209
MLCOUNT() 210
MLPOS() 210
MOD()* 211
MONTH() 211
NETERR() 211
NETNAME() 212
NEXTKEY() 212
OS() 212
OUTERR() 212
OUTSTD() 212
PAD() 213
PCOL() 213
PCOUNT() 213
PROCLINE() 214
PROCNAME() 214
PROW() 214
QOUT()|QQOUT() 214
RAT() 215
READEXIT() 215
READINSERT() 215
READKEY()* 216
READMODAL() 216
READVAR() 216
RECCOUNT()* 217
RECNO() 217
RECSIZE() 217
REPLICATE() 217
RESTSCREEN() 218
RIGHT() 218
RLOCK() 218
ROUND() 219
ROW() 219
RTRIM() 219
SAVESCREEN() 219
SCROLL() 220
SECONDS() 220
SELECT() 220
SET() 221
SETBLINK() 221
SETCANCEL() 221
SETCOLOR() 221
SETCURSOR() 222
SETKEY() 222
SETMODE() 222
SETPOS() 223
SETPRC() 223
SOUNDEX() 223
SPACE() 223
SQRT() 224
STR() 224
STRTRAN() 224
STUFF() 225
SUBSTR() 225
TIME() 226
TONE() 226
TRANSFORM() 226
TYPE() 226
UPDATED() 227
UPPER() 227
USED() 227

VAL() 228
VALTYPE() 228
VERSION() 228
WORD()* 229
YEAR() 229

Osztályok

Hibakezelő 251
Adatbeviteli 253
Adatáttekintési 255
Adatmegjelenítési 257

Paraméterek (szerkesztési)

BATCH 243
BEGINAREA...ENDAREA 247
DEBUG 244
DEFAULTLIBRARYSEARCH 246
DYNAMIC 244
EXCLUDE 246
EXTDICTIONARY 246
FREEFORMAT 243
HELP 244
IGNORECASE 243
INCREMENTAL 245
MAP 244
NOBATCH 243
NODEFAULTLIBRARYSE-
ARCH 246
NOEXTDICTIONARY 246
NOIGNORECASE 243
NOINCREMENTAL 245
POSITIONAL 243
PRELINK 247
REFER 247
RESIDENT 244
SILENT 245
STACK 246
STATIC 247
VERBOSE 245

Parancsok

?|?? 69
@...BOX 70
@...CLEAR 71
@...GET 71
@...PROMPT 74
@...SAY 75
@...TO 77
ACCEPT 78
APPEND BLANK 78
APPEND FROM 79
AVERAGE 80
CALL* 81
CANCEL* 82
CLEAR ALL* 83
CLEAR GETS 83
CLEAR MEMORY 83
CLEAR SCREEN 84
CLEAR TYPEHEAD 84
CLOSE 84
COMMIT 85
CONTINUE 86
COPY FILE 87
COPY STRUCTURE 87
COPY STRUCTURE EXTENDED
88
COPY TO 89
COUNT 91
CREATE 92
CREATE FROM 92
DELETE 94
DELETE FILE 95
DIR* 95
DISPLAY 96
EJECT 97
ERASE 98
FIND* 99
GO 99
INDEX 101
INPUT 102
JOIN 103
KEYBOARD 104
LABEL FORM 105
LIST 106
LOCATE 107
MENU TO 109
NOTE* 110
PACK 111
QUIT 112
READ 112
RECALL 114
REINDEX 115
RELEASE 116
RENAME 116
REPLACE 117
REPORT FORM 118

RESTORE 120
RESTORE SCREEN* 121
RUN 122
SAVE 123
SAVE SCREEN* 123
SEEK 124
SELECT 125
SET ALTERNATE 126
SET BELL 128
SET CENTURY 128
SET COLOR* 129
SET CONFIRM 131
SET CONSOLE 131
SET CURSOR 132
SET DATE 133
SET DECIMALS 134
SET DEFAULT 135
SET DELETED 135
SET DELIMITERS 136
SET DEVICE 137
SET EPOCH 138
SET ESCAPE 138
SET EXACT* 139
SET EXCLUSIVE* 140
SET FILTER 141
SET FIXED 142
SET FORMAT* 142
SET FUNCTION 143
SET INDEX 144
SET INTENSITY 145
SET KEY 145
SET MARGIN 147
SET MESSAGE 147
SET ORDER 148
SET PATH 149
SET PRINTER 150
SET PROCEDURE* 151
SET RELATION 152
SET SCOREBOARD 153
SET SOFTSEEK 154
SET TYPEHEAD 155
SET UNIQUE* 155
SET WRAP 156
SKIP 156
SORT 158
STORE* 159
SUM 160
TEXT* 161
TOTAL 162
TYPE 163
UNLOCK 164
UPDATE 165
USE 166
WAIT* 168
ZAP 169

Utasítások

BEGIN SEQUENCE 35
DECLARE* 37
DO* 37
DO CASE 39
DO WHILE 40
EXTERNAL 41
FIELD 42
FOR 43
FUNCTION 44
IF 47
LOCAL 48
MEMVAR 49
PARAMETERS 50
PRIVATE 51
PROCEDURE 52
PUBLIC 54
RETURN 56
STATIC 57

A kézikönyv a Nantucket Clipper 5.0 dokumentációinak felhasználásával készült.

A Nantucket® és a Clipper® a Nantucket Corporation védjegye
A dBase®, dBase III® és a dBase III PLUS™ az Ashton-Tate Corporation védjegye
Az IBM® és a PC-DOS® az International Business Machine védjegye
Az Intel® az Intel Corporation védjegye
A Microsoft® és az MS-DOS® a Microsoft Corporation védjegye
Az .RTLINK™, © Pocket Soft, Inc., 1988, 1989. a Pocket Soft Inc. védjegye

1991. év könyvújdonságai

Donáth János
IBM PC INFORMÁCÓSKÁRTYA 187.-Ft

Pajor Gábor
IBM PC-RÓL ALAPFOKON A SZOFTVER 101.-Ft
IBM PC-RÓL ALAPFOKON A HARDVER 100.-Ft

Pirkó József
TURBO PASCAL 5.5 499.-Ft

PROGRAMOZÁSI SEGÉDSZOFTVEREK IBM PC-RE (DOS, dBASE III+, CLIPPER)
Dr. Dedinszky Ferenc
SAYWHAT, DBU, DEBUG 184.-Ft
CLIPPER SEGÉDSZOFTVEREK 210.-Ft

Fekete - Nagy - Gregorics
BEVEZETÉS A MESTERSÉGES INTELLIGENCIÁBA 274.-Ft

A 32 BITES MIKROPROCESSZOROK
Dr. Kovács Magda
80386 MIKROPROCESSZOR I. 189.-Ft

Dr. Kovács Magda
MIKROELEKTRONIKAI ÉRTELMEZŐSZÓTÁR I-III. 699.-Ft

Dr. Dedinszky Ferenc
CLIPPER 5.0 KÉZIKÖNYV 340.-Ft

Gál I. - Dallos E.:
QUICK BASIC * lemezmelléklettel 369.-Ft

Benkő Tiborné - Poppe András - Benkő László
TURBO C++ * lemezmelléklettel 435.-Ft

Németh János
TURBO C 2.0 FEJLESZTŐRENDSZER irányár: 300.-Ft

Bartha Attila
NORTON (átdolgozott kiadás) irányár: 200.-fT

Nyéki L. - Nagy T.
TURBO BASIC irányár: 300.-Ft

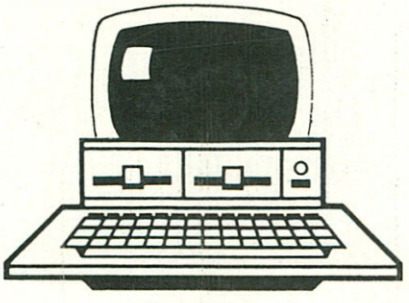
Kiadványok megrendelhetőek, ill. megvásárolhatóak:
LSI SHOP
1077 Bp., Király u. 91.
Tel.: 12-21-076

Viszonteladókat várjuk:
LSI Informatikai Osztály
1126 Bp., Tartsay V. u. 12.
Tel.: 17-51-564

Ára: 340,- Ft

SZOinfo

NSZK — MAGYAR
INFORMATIKAI ÉS KERESKEDELMI KFT.



SZÁMÍTÁSTECHNIKAI, ÜGYVITELI
ÉS IRODATECHNIKAI ESZKÖZÖK
BESZERZÉSE, FORGALMAZÁSA

KOMPLEX RENDSZERKIDOLGOZÁS:
- — RENDSZERSZERVEZÉS
- — PROGRAMKIDOLGOZÁS
- — OKTATÁS
- — BETANÍTÁS

KOMPLEX SZERVIZTEVÉKENYSÉG

EXKLUZÍV HALOGÉN VILÁGÍTÁS-
TECHNIKAI ESZKÖZÖK ÁRUSÍTÁSA
- — LAKÓTÉRI
- — IRODAI
- — KÜLSŐ TÉRI

Központ: 1118 Budapest, Bozókvár u. 11
Tel.: (36-1) 18-12-646; Fax: (36-1) 16-65-413

Üzlet: Budapest, Király u. 59/B
Tel.: (36-1) 14-22-059

Bemutatóterem: Budapest, Gazdagréti tér 7.
Tel.: (36-1) 17-36-637