

Magiszter

Számítástechnikai szerkesztőség

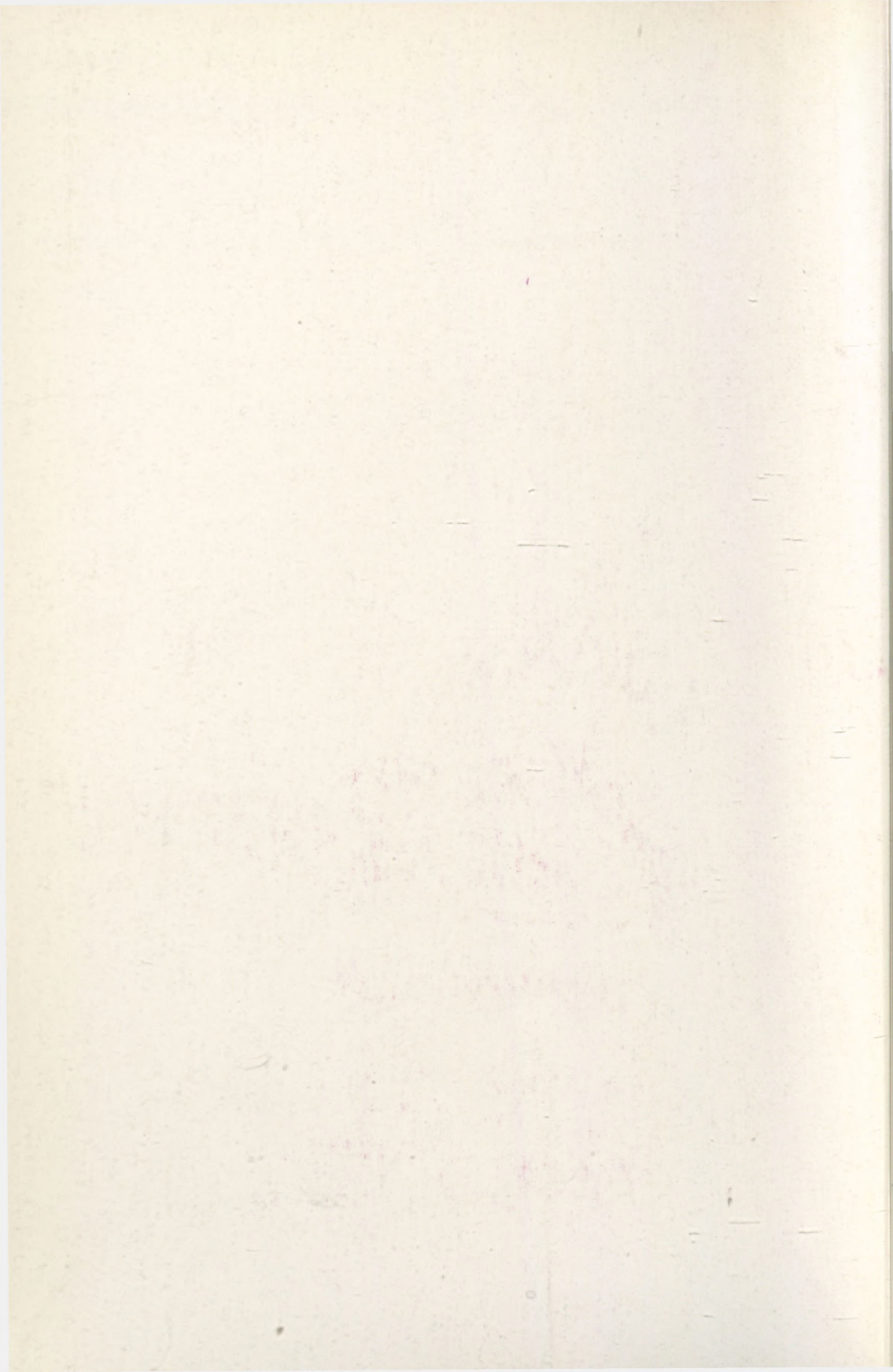
MFORM

Adatbeviteli alapszoftver és képernyőkezelőrendszer

Programvédelem nélkül!

Szoftver és
IBM, PC, XT, AT
kompatibilis mikroszámítógépekre





2676
GONDOLA-KISS-KRAFFT-SCHILL-SZEGEDI

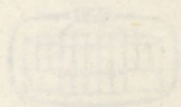
M-FORM

Adattípus formagenerátor

M-FORM

Adattípus formagenerátor és feltöltő rendszer

MS C 5 XX környezetben



Akadémiai Kiadó, Budapest 1988

NTAK



9 50040 32796 9

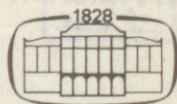
23676

GONDOLA ZOLTÁN-KISS GYÖRGY-KRAFFT WALTER
SCHILL RÓBERT-SZEGEDI JÁNOS

M-FORM

Adattípus formagenerátor és futtató rendszer

MS C 5.XX környezetben



Akadémiai Kiadó, Budapest 1989

MTAK



0 00002 32796 9

5544/89.

668896

Szoftver - dokumentáció

Sorozatszerkesztő: Pintér Tibor

A kiadványt szakmailag ellenőrizte: Benedikti István

©Akadémiai Kiadó és Nyomda Vállalat

GONDOLA-KISS-KRAFFT-SCHILL-SZEGEDI, 1989

Minden jog fenntartva

MAGYAR
TUDOMÁNYOS AKADÉMIA
KÖNYVTÁRA

M. TUD. AKADÉMIA KÖNYVTÁRA
Könyvtár 60.6.6. /19...89. sz.

A szoftver - dokumentáció a $\text{M}\text{T}\text{E}\text{X}$ programcsomaggal készült

©Akadémiai Kiadó és Nyomda Vállalat · Fadgyas – Miklós

Tartalomjegyzék

1. Bevezetés	7
2. Általános ismertetés	9
3. A form felépítése	13
3.1. A form-file	13
3.2. A FORM= sor	14
3.3. A TEXT= sor	15
3.4. A GRAPH= sor	15
3.5. A FIELD= sor	16
3.6. Az ASSKEY= sor	17
4. A FormEdit	18
4.1. Használt file-ok	19
4.2. Hívása	19
4.3. Kiszállítás	20
4.4. Általános parancsok	21
4.5. Pozícionálás	21
4.6. A képernyőn elfoglalt hely	22
4.7. Többblapos adatkezelés	23
4.8. Színek állítása	23
4.9. Mező létrehozása	24
4.10. Mező törlése	25
4.11. Mező módosítása	25

4.12. SUBFORM mező létrehozása	28
4.13. Szövegkonstans létrehozása	30
4.14. Szövegkonstans törlése	31
4.15. Szövegkonstans módosítása	31
4.16. Státusz sor beállítása	31
4.17. Grafikus elem létrehozása	32
4.18. Grafikus elem törlése	32
4.19. Grafikus elem módosítása	32
4.20. Mozgatás	33
4.21. Mezők és szövegkonstansok másolása	33
4.22. A mezők áttekintő listája	33
4.23. Az aktuális mező jellemzői	34
4.24. Billentyűk elrendelése	34
4.25. Aktuális képernyőpozíció lekérdezése	34
4.26. A képernyő újraírása	35
4.27. Kimentés file-ba	35
4.28. Programrészek generálása	35
4.29. Kilépés az operációs rendszerbe	38
4.30. A parancsok összefoglalása	38
5. A programozás	40
5.1. A futtatórendszer függvényei	40
5.2. Felhasználói függvények	80
5.3. Keretprogram	97
5.4. Egyszerű példaprogram	98
5.5. Teljes példa	103
5.6. Hibakezelés	164
6. A FormDoc	165
7. Operátori kezelés	166
8. A disztributív lemez tartalma	168

1. fejezet

Bevezetés

A számítógépes adatfeldolgozó programok általában adatbázisokon dolgoznak. Az adatbázisok feltöltése, módosítása, lekérdezése és egyéb módon való kezelése a számítógép és az operátor közötti kommunikációval valósul meg. Korunkban ez a kommunikáció elsősorban teljes képernyős formában, az adott számítógép képernyője és billentyűzete által kínált lehetőségek minél magasabb szintű kihasználásával, az operátori igények messzemenő figyelembevételével képzelhető el.

Az adatbeviteli képernyők definiálására és programozására számos ismert szoftver használható:

FORM (FMS, RSX 11 öp.r.)

Hewlett Packard FORM/250

DiaGen (IBM 370)

Mcis (Unix)

VITAMIN C

Ztools

dBASE

ORACLE

Progress

A fenti rendszerek különböző számítógépeken, más és más operációs rendszer alatt működnek, de feladatuk többé kevésbé hasonló: segíteni az adatkezelő program és az operátor közötti párbeszédet.

A felsorolt rendszerek ezt a segítséget különböző szemszögből, más-más szinten és mélységben adják. Egyesek integrált rendszeren belül működnek, szorosan kapcsolódva az adatbáziskezelő rendszerhez, míg mások általános kommunikációs segédeszköznek tekinthetők.

Számunkra több éves gyakorlat és számos adatfeldolgozó rendszer elkészítése után vált egyértelművé, hogy a viszonylag nagy kínálat mellett is szükség van egy PC környezetben működő, C nyelvhez kapcsolódó, saját rendszer létrehozására, amely leveszi a programozó válláról a párbeszéd megszervezésének, programozásának és karbantartásának terhét. Emellett gazdag illeszkedési választékával gyakorlatilag korlátlan felügyeleti lehetőséget biztosít a programozó számára az adatbevitel menete felett.

Az M-FORM egy ilyen adatkezelési segédeszköz, amely a C nyelven adatfeldolgozó programokat író szoftverek számára készült.

A kézikönyv feltételezi az olvasóról, hogy tisztában van az alkalmazói szoftverek felépítésével, adatbázisok kezelésével és az IBM PC/XT/AT DOS környezetében van megfelelő C nyelvi gyakorlata.

2. fejezet

Általános ismertetés

Az M-FORM több (kész és készülő) integrált rendszer része, így különböző módokon illeszkedik

- az ADAPT alkalmazói programszervező rendszerhez (Magiszter Software 1989),
- a MISAM index-szekvenciális file-kezelő rendszerhez (Magiszter Software 1988),
- a CEX és a CEX2 C könyvtárakhoz (Magiszter Software 1987, 1988),
- a DOG szövegformázóhoz (Magiszter Software, 1987).

Az M-FORM ezenkívül önállóan is használható a C programozók számára, saját file-kezelő stb. rendszereket feltételezve.

A rendszer alkotóelemei:

1. A FormEdit, amely kényelmes és hatékony eszköz egy adatbeviteli formátum létrehozására, módosítására és számos generatív funkcióval segíti a programozói munkát.
2. A futtató rendszer, amely C könyvtárak formájában, Microsoft C, Borland TurboC, QuickC és Aztec C nyelvek különböző memóriamodelljeihez létezik.
3. A Form Documenter, amely alkalmas egy adatbeviteli formátum jól használható dokumentációjának elkészítésére.

Az adatkezelés módja az M-FORM rendszerrel a következő:

A programozó által megírt keretprogram végzi az adatbázis és a form rendszer megnyitását, lezárását, és az adatbevitel menetétől függően felkészíti a form rendszert az aktuális rekord kezelésére, majd átadja a vezérlést az M-FORM futtatórendszerének.

A futtatórendszer elvégzi az összes képernyő- és billentyűkezelési feladatot, lehetőséget ad az egyes rekordelemek definiálására, módosítására. A programozó által megvalósított funkciókat végző C függvények a vezérlést és a

megfelelő paramétereket a futtatórendszerrel kapják. Ezek a C függvények visszatérésük módjával befolyásolhatják a futtatórendszer további működését. A futtatórendszerből való operátori kilépés után a keretprogram folytathatja az adatfeldolgozást.

Az M-FORM használatának előfeltételei:

- IBM PC/XT/AT vagy kompatibilis számítógép,
- MS-DOS/PC-DOS 3.xx operációs rendszer,
- 256 Kbyte RAM,
- a következő C nyelvi környezetek valamelyike:
 - Microsoft C 4.xx, 5.xx,
 - Borland TurboC 2.xx,
 - QuickC 5.x,
 - Aztec Manx C 3.xx, 4.xx,
- a CEX és a CEX2, C könyvtárak megfelelő változatai (Magszter Software).

A továbbiakban a következő speciális fogalmakat használjuk:

form

az az információ halmaz, amely a futtató rendszer számára egy adatbeviteli folyamat minden információját tartalmazza.

form-file

egy form leírását kötött, de szerkeszthető formában tartalmazó file.

form-elem

a form alapeleme, amely lehet mező, szövegkonstans vagy grafikus elem.

mező

az a form-elem, amely egy adat bevitelére vagy kiírására szolgál.

input-mező

az a mező, amely egy adat bevitelére szolgál.

output-mező

az a mező, amely egy adat kiírására szolgál.

aktuálismező

az az input mező, amelyen az operátori kezelés során a kurzor áll.

szövegkonstans

a képernyőre kikerülő szöveg, amely az egyes mezők értelmezését segíti, vagy általános információkkal szolgál az operátor számára.

státuszszor

olyan speciális szövegkonstans, amelynek tartalma mindig az aktuális mezőhöz rendelt magyarázó szöveg.

grafikus elem

a karakterkészlet kvázi-grafikus elemeiből összeálló vízszintes vagy függőleges vonal, ill. keret, amely a képernyőn látható információk csoportosíthatóságát, ill. elkülönítését segíti.

USER mező

olyan tetszőleges típusú mező, amely nem az M-FORM által támogatott elemi C típusú, hanem egyéb a programozó által kezelt, sajátos mező.

alrekord

a form által kezelt rekordhoz tartozó változó számú adatsor egy eleme.

al-form

egy olyan form, amely egy alrekord kezelésének információit tartalmazza (szerkezetében megegyezik a form szerkezetével).

SUBFORM mező

olyan mező, amelyhez egy al-form van rendelve, és a mező és az al-form méretétől függően egy vagy több alrekord kezelését teszi lehetővé.

felhasználói függvény

olyan, a programozó által megírt (megírandó) C függvény, amely lehet

- beolvasó függvény,
- ellenőrző függvény,
- kiíró függvény,
- betöltő függvény,
- kimentő függvény,
- elrendelt függvény,

beolvasó függvény

olyan C függvény, amely a mező értékének beolvasását nem az M-FORM által támogatott módon valósítja meg.

ellenőrző függvény

olyan C függvény, amely az adott mező módosítása után kapja a vezérlést és ellenőrzési vagy egyéb adatkezelő algoritmust hajt végre.

kiíró függvény

olyan C függvény, amely a USER mező megjelenítését végzi.

betöltő függvény

olyan C függvény, amely a SUBFORM mezőn belül az aktuális alrekord betöltését végzi.

kimentő függvény

olyan C függvény, amely a SUBFORM mezőn belül az aktuális alrekord kimentését végzi (módosítás vagy beszúrás után).

elrendelt függvény

olyan függvény, amely egy meghatározott billentyű leütésére, a billentyűhöz rendelt funkciót valósítja meg.

elrendelt billentyű

olyan billentyű, amelyhez egy felhasználói függvény van rendelve.

3. fejezet

A form felépítése

Egy adatbeviteli form a következő részekből épül fel:

- a form általános adatai,
- a szövegkonstansok leírása,
- a grafikus elemek leírása,
- a mezők leírása,
- az elrendelt függvények leírása,

Ezeket az információkat egy kötelezően FTX kiterjesztésű, ún. form-file tartalmazza.

3.1. A form-file

A form definiálása az adatkezeléshez szükséges összes információnak a rögzítése egy form-file-ba. Ezt a form-file-t a keretprogramban egy utasítással beolvashatjuk, és ettől kezdve a futtatórendszer képes a form által meghatározott adatkezelésre.

A form-file egy kötött formájú szövegfile, amelynek létrehozása és módosítása nagy körütekintést igényel, mivel az egyes jellemzők között bonyolult összefüggések vannak. Ezért a form-file karbantartását a FormEdit editorral célszerű végezni, amellyel kényelmesen és ellenőrzöttén kezelhető a form-file és gazdag szolgáltatásai számottevően megkönnyítik a programozási feladatokat is.

A form-file egy vagy több form információit tartalmazza kötött, de editálható formában. Ha egy form-nak vannak SUBFORM mezői, akkor a SUBFORM mezőhöz tartozó form-ok leírásainak szintén szerepelniük kell az adott form-file-ban.

A file-ban az egyes információk kulcsszavas sorokban vannak. Egy form leírása a FORM= kulcsszóval kezdődő sortól a file-végéig vagy a következő FORM= kulcsszóig tart. A file-ban a következő kulcsszavak értelmeződnek:

- FORM=
- TEXT=
- GRAPH=
- FIELD=
- ASSKEY=

A FORM= kulcsszó egy form leírásában természetesen csak egyszer szerepel, a többi annyiszor, ahány elem tartozik a form-hoz. Az elemek sorrendje tetszőleges.

3.2. A FORM= sor

A FORM= sor egy form általános jellemzőit írja le. A form további elemeit (szövegkonstansok, grafikus elemek, mezők és elrendelt billentyűk) a további sorok írják le a következő FORM= sorig, ill. a form-file végéig.

Formája:

FORM= n,d,ul:lc-ll:rc,bf/bb,hf/hb,ef/eb,mf/mb,if/ib

ahol:

- n (name) a form neve (maximum 8 hosszúságú alfanumerikus azonosító);
- d (description) a form funkciója (max. 80 hosszúságú tetszőleges karaktersor);
- ul (upper line) a form felső sora;
- lc (left column) a form bal oszlopa;
- ll (lower line) a form alsó sora;
- rc (right column) a form jobb oszlopa;
- bf (base color foreground) az alapszín előtér színe;
- bb (base color background) az alapszín háttér színe;
- hf (highlighted color foreground) a kiemelt szín előtér színe;
- hb (highlighted color background) a kiemelt szín háttér színe;
- ef (editing color foreground) a szerkesztő szín előtér színe;
- eb (editing color background) a szerkesztő szín háttér színe;
- mf (message color foreground) az üzenetszín előtér színe;
- mb (message color background) az üzenetszín háttér színe;

- if (help info color foreground) a helpszín előtterszíne;
- ib (help info color background) a helpszín hátterszíne.

3.3. A TEXT= sor

A TEXT= sor egy szövegkonstanst ír le.

Formája:

TEXT= l,c,cf/cb,t

ahol:

- l (line) a szöveg sora;
- c (column) a szöveg oszlopa;
- cf (color foreground) a szöveg előtterszíne;
- cb (color background) a szöveg hátterszíne;
- t (text) a szöveg, amely sor végéig tart.

3.4. A GRAPH= sor

A GRAPH= sor egy grafikus elemet ír le.

Formája:

GRAPH= ul:lc-ll:rc,t,cf/cb

ahol:

- ul (upper line) az elem felső sora;
- lc (left column) az elem bal oszlopa;
- ll (lower line) az elem alsó sora;
- rc (right column) az elem jobb oszlopa;
- t (type) az elem típusa (HDVD,HSVS,HSVD,HDHS), lásd a CEX <form.h> header-file-ját;
- cf (color foreground) az elem előtterszíne;
- cb (color background) az elem hátterszíne.

3.5. A FIELD= sor

A FIELD= sor egy mező leírását adja meg.

A mező leírásában típustól függően egyes elemek mást és mást jelenthetnek.

Formája:

FIELD= n,d,s,t,pfd,l,c,p1,p2,if,uf,def

ahol:

- n (name) a mező neve (max. 15 hosszúságú alfanumerikus azonosító);
- d (description) a mező funkciója (max. 80 hosszúságú tetszőleges karaktersor);
- s (status) a mező státusz sora (max. 80 hosszúságú tetszőleges karaktersor), amely a mezőre való rálépéskor kerül kiírásra az ún. státusz sorban;
- t (type) a mező típusa (BYTE, INT, LONG, FLOAT, DOUBLE, DATE, TEXT, USER, SUBFORM);
- pfd (Protected, Formatted, Default) output (védett) mező, formátumos mező, alapértelmezett értékkel ellátott mező (a megfelelő tulajdonságot a kezdőbetű megléte jelenti);
- l (line) a mező sora;
- c (column) a mező oszlópa;
- p1 ha a mező USER vagy SUBFORM típusú, akkor a mező alsó sora, egyébként a mező hossza;
- p2 ha a mező egész (BYTE,INT, LONG) típusú, akkor a 0-tól különböző érték azt jelzi, hogy a mező előjel nélküli, ha a mező lebegőpontos (FLOAT,DOUBLE) típusú, akkor a decimális jegyek száma, ha a mező DATE típusú, akkor a dátum formája (0: angol, 1: magyar, 2: német), ha a mező USER vagy SUBFORM típusú, akkor a mező jobb oszlópa;
- if (input function) beolvasó (SUBFORM-nál betöltő) függvény neve, ill. , ha a mező formátumos, akkor a formátum karaktersorozata;

- uf (update function) ellenőrző (SUBFORM-nál kiíró USER-nél megjelenítő) függvény neve;
- def (default) a mező alapértelmezett értéke.

3.6. Az ASSKEY= sor

Az ASSKEY= sor egy elrendelt függvényt, azaz egy billentyűelrendelést ír le.

Formája:

ASSKEY= kn,fn

ahol:

- kn (key name) a billentyű szabványosított neve;
- fn (function name) a hozzárendelt C függvény neve.

4. fejezet

A FormEdit

Egy adatkezelő form a FormEdit teljes képernyős szerkesztő programmal definiálható.

A FormEdit szolgáltatásai:

- A form képernyőn elfoglalt helyének beállítása.
- A form rövid funkcióleírásának megadása.
- A form-hoz rendelt öt szín (alapszín, input-mező szín stb.) beállítása.
- Mezők definiálása, ill. módosítása.
- Szövegkonstansok definiálása, ill. módosítása a kívánt színnel.
- Grafikus elemek definiálása, ill. módosítása a kívánt színnel.
- A form-elemek helyének változtatása.
- Mezők és szövegkonstansok másolása.
- A már definiált mezők jellemzőinek listázása a képernyőre.
- A billentyűkhöz elrendelt függvények rendelése.
- A definiált form kimentése file-ba.
- A form-hoz használt felhasználói függvények táblájának (C forrás) generálása.
- A form-hoz C programba beszúrandó header-file generálása.
- A felhasználói C függvények keretének generálása.
- A form-ot dinamikusan létrehozó C forrásmodul generálása.
- A teljes form és a definiált mezők jellemzőinek listázása.
- Felhasználói útmutatás kérése.
- Kilépés a DOS operációs rendszerbe.

4.1. Használt file-ok

A FormEdit inputja egy korábban létrehozott form-file lehet, amelynek típusa alapértelmezés szerint .FTX.

A FormEdit output file-jai:

1. Egy form-file, amely a párbeszéd során rögzített form-file információkat tartalmazza
 - a futtatórendszer számára és
 - a későbbi módosítás számára.
2. Egy C nyelvű forrásmodul, amely a felhasználói programmal összeszerkesztendő felhasználói függvények tábláját definiálja.
3. Egy definíciós C forrás (header-file), amely a form egyes mezőinek sorszámát definiálja, valamint a form mezőinek a belső bufferét lefedő struktúra-definíciót tartalmaz. A header-file a futtatórendszer rutinjainak használatához szükséges, mivel a rutinok a mező sorszáma alapján érik el az egyes mezőket. A struktúra az M-FORM hatáskörén kívül eső (számítási) feladatok elvégzését segíti.
4. A felhasználói C függvények keretét (definíció, paraméterek, kommentek, visszatérési érték) tartalmazó C forrás-file.
5. A form-ot dinamikusan létrehozó C forrásmodul.

A felsorolt öt file neve és típusa kötött:

a név megegyezik a form nevével,
a file-típus pedig rendre:

.FTX	(Form TeXt),
.FCT	(Form C function Table),
.FH	(Form Header),
.FCC	(Form C function C source),
.FIC	(Form Initializing C source).

4.2. Hívása

A FormEdit hívása a következő:

```
>FORMEDIT [-i] [-b] [-o headerpath] form-file
```


Ahol:

<i>form-file</i>	a korábban létrehozott vagy még nem létező form-file;
-i	a bejelentkező képernyő letiltása;
-b	fekete-fehér üzemmód;
<i>headerpath</i>	a generálandó header-file és függvény- definíciós modulok könyvtára (ha az eltér a <i>form-file</i> könyvtárától).

Indítás után

- a form megjelenik a képernyőn,
- új form esetén az üres form az alapértelmezett méretben és színekkel,
- korábban létrehozott form esetén a már definiált form,
- a kurzor a form képernyőrészének bal felső sarkába áll és
- a rendszer várja a parancsokat.

4.3. Kiszállítás

A rendszerből való kiszállást az ESC billentyű lenyomásával lehet kezdeményezni.

Ha a form leírásában az indítás, vagy az utolsó mentés óta nem volt változás, akkor a FormEdit befejezi a működését.

Ha a form leírásában az indítás, vagy az utolsó mentés óta volt változás, akkor a FormEdit működése a következő:

- A rendszer felajánlja a form információinak kimentését és a header-file generálását. A file-ba történő kiírás az igenlő válasszal végezhető el. Ha a header-file generálását kértük, és már létezik a form-ot dinamikusan létrehozó C forrásmodul (.FIC file), akkor ennek generálását is felajánlja a rendszer.
- Ha a form-hoz tartozó függvények változtak, akkor a rendszer felajánlja a felhasználói függvények táblájának generálását. A kérdésre adott igenlő válasz esetén a *formnév.FCT* nevű file-ban keletkezik a megfelelő tábla. Ha az FCT file generálását kértük, akkor a rendszer a felhasználói C függvények keretét tartalmazó C forrás-file-ra (.FCC) is felajánlja a mentést.

Ha a kiírások sikeresek voltak vagy az adott file felülírását visszautasítottuk, akkor a FormEdit befejezi a működését.

Ha form mentése írási hiba (pl. tele a lemez) miatt nem volt sikeres, akkor a rendszer nem száll ki, tovább várja a parancsokat, így a DOS operációs rendszerbe való kiszállás után elháríthatjuk a hiba okát.

4.4. Általános parancsok

A FormEdit funkciói általában egy billentyű leütésével aktivizálhatók. Az Alt+*betű* billentyűkombinációból a *betű* a funkció angol megfelelőjére utal, a kurzorvezérlő billentyűk használata értelemszerű.

Módosítások esetén az ENTER billentyű az aktuális jellemzők elfogadását, az ESC billentyű pedig a módosítás előtti állapot visszaállítását jelenti.

A rendszer által feltett?(y/n) típusú kérdésekre

az 'Y' és 'y' (YES) billentyűkkel adhatunk igen

a 'N' és 'n' (NO) és ESC billentyűkkel adhatunk nemleges választ.

Több helyen találkozunk úgynevezett kiválasztási menüvel, amelyből a következő három módon választhatunk:

1. pozícionálás a kívánt tételre és az ENTER billentyű leütése,
2. a kívánt tétel nevében található nagybetű leütése (nem mindig az első betű!),
3. az ESC billentyű leütése a választás visszautasítása.

4.5. Pozícionálás

A kurzort a billentyűzet úgynevezett numerikus billentyűzeténél található kurzorvezérlő billentyűkkel lehet pozícionálni:

- A kurzor mozgatása egy pozícióval:

Up, Down, Left, Right, Space.

A Space (szóköz) billentyű pozícionáláskor egyenértékű a Right billentyűvel.

- A kurzor mozgatása az aktuális sor elejére vagy végére:

Home, End.

- A kurzor mozgatása az aktuális oszlop tetejére vagy aljára:
PgUp, PgDn.
- Pozícionálás az előző mezőre:
^Left.
- Pozícionálás a következő mezőre:
^Right.
- Pozícionálás az első mezőre:
^Home.
- Pozícionálás az utolsó mezőre:
^End.

Ha a kurzor egy mező vagy egy szövegkonstans első pozícióján áll, akkor az egy pozícióval való jobbra mozgatással a mező, ill. szövegkonstans utáni (nem foglalt) pozícióra lehet ráállni.

Mezőt, szövegkonstanst vagy grafikus részt annak első pozíciójára való ráállással lehet kiválasztani.

A képernyő egy területét (USER és SUBFORM mező definiálása és mezők ill. szövegkonstansok mozgatása esetén) a kurzorvezérlő billentyűkkel lehet kijelölni:

- Left, Right, Up, Down: a kijelölt terület növelése vagy csökkentése a kívánt irányba egy pozícióval,
- Home, End, PgUp, PgDn: a kijelölt terület növelése vagy csökkentése a kívánt irányba a lehetséges leg szélső értékre.

4.6. A képernyőn elfoglalt hely

A form méretének és helyének beállítása az Alt-W (Window) billentyű leütésével kezdeményezhető. (Ez a funkció csak a form legelső oldalán kérhető!) Ekkor a képernyő jobb felső sarkában egy kiválasztási menü jelenik meg 2 vagy 3 lehetőséggel:

1. Move window
2. Change size
3. Select actual box as frame

A harmadik lehetőség csak akkor jelenik meg, ha a kurzor egy grafikus elem bal felső sarkán áll (tehát ez az aktuális form-elem), és az elem átal behatárolt négyszögön kívül nincs form-elem.

A **Move window** választása után a kurzor-billentyűk segítségével (Up, Down, Left, Right, Home, End, PgUp, PgDn) a teljes form-ot a kívánt irányba elmozgathatjuk. Az ENTER billentyű leütésével a form helyét rögzítjük, ill. az ESC billentyűvel az elmozgatást törölhetjük.

A **Change size** választása után a kurzor-billentyűk segítségével a form alsó (Up, Down, PgUp, PgDn) és jobb (Left, Right, Home, End) szélét a kívánt irányba, a lehetséges mértékig szűkíthetjük, ill. bővíthetjük. Ha a form alsó szélét úgy húzzuk fel, hogy az új méretű form első lapjára egyes mezők nem férnek el, akkor ezen mezők automatikusan a következő lapra kerülnek. A form mérete nem csökkenthető le annyira, hogy egy SUBFORM típusú mező a form második lapjára kerüljön.

A méretcsökkentéssel létrehozhatunk egy olyan többlapos form-ot is, amelynek egy lapján nincs 25 sor.

Az aktuális méretet az ENTER leütésével rögzíthetjük, ill. az ESC billentyűvel a változtatást törölhetjük.

A **Select actual box as frame** választása esetén a form mérete a négyzet által behatárolt terület lesz.

A mozgatás vagy méretmódosítás után a képernyőn a form területén kívül maradhat "szemét" (átszínezett terület). Ezt az Alt-R (Redisplay) billentyű leütésével eltüntethetjük.

4.7. Többlapos adatkezelés

Az adatfeldolgozási programoknál előfordul, hogy az egy rekordhoz tartozó mezők nem férnek el egy képernyőn. Ilyenkor többlapos form-ot kell létrehozunk. Ez a kurzorvezérlő billentyűkkel, egyszerűen a következő lapra való pozicionálással tehető meg. (Lásd még a 4.6. **Change size** részét!)

4.8. Színek állítása

A form színeit az Alt-C (Colors) billentyű leütése után állíthatjuk be.

A form 5 színt kezel (a színek magyarázata a futtatási fázisra vonatkozik):

base	alapszín és az output mezők színe,
highlighted	az input mezők színe,
edit	az input mezők változtatási színe,
message	az üzenet és a hibajelzés színe,
help	a segítő információk színe.

A megjelenő színtáblázaton a kurzorvezérlő billentyűkkel ki kell választani a változtatandó színt, majd a + és - billentyűkkel a kívántra beállítani. Minden színnél külön állítható a háttér- és a kiírás színe.

A módosítást törölni lehet az ESC billentyű leütésével. A megfelelő palettát, beállítása után azt az ENTER billentyűvel rögzíthetjük.

Ha a form-ban al-formok is vannak, vagy éppen egy al-form definiálunk, akkor a "Copy colors to other forms?" kérdésre adott igenlő válasszal az adott színkombinációt a többi form-ba is átmásolhatjuk (pl.: fekete-fehér képernyőre való áttérés esetén).

Ha az alapszínt változtattuk meg, akkor a form szövegkonstansainak és grafikus részeinek színét az új alapszínhez igazíthatjuk a következő kiválasztási menü alapján:

Change the color change only	a megegyező színűek átszínezése,
Foreground color change only	a megegyező kiírási színűek átszínezése,
Background color	a megegyező háttérszínűek átszínezése.
Használatuk értelemszerű.	

4.9. Mező létrehozása

A mező beszúrását az aktuális pozícióra az INS billentyű lenyomása után megjelenő kiválasztási menü első tételének (Field) kiválasztásával kezdjük.

A mező meghatározása a mező nevének megadásával kezdődik: az "Enter the name of field:" kiírás után meg kell adni a maximum 15 karakter hosszú, C szabályoknak megfelelő nevet.

Ekkor a mező definiálttá válik a következő jellemzőkkel:

- előjeles byte típusú,
- input mező,
- hossza 1 karakterpozíció.

Ezután a mező-módosító ablak kerül kiírásra, ahol a mező egyes jellemzőit a kívánt értékre beállíthatjuk (Lásd a 4.11. Mező módosítása c. részt!)

4.10. Mező törlése

A mezőt, a mezőre való pozicionálással, majd a DEL billentyű leütésével lehet törölni. A törlési szándékot meg kell erősíteni az "Are You sure to delete ... field?" kérdésre adott igenlő válasszal.

4.11. Mező módosítása

Az aktuális mező módosítása az INS billentyű leütésével kérhető. Hatására egy 10 elemű, 0-9-ig számozott táblázat jelenik meg, amelynek egyes tételei a mező jellemzőit tartalmazzák. Egyes tételeknél a tartalom a mező típusától függ, ezekről minden típushoz leírjuk a megfelelő magyarázatot. A kívánt jellemzőt a tételre való ráállás és az ENTER billentyű leütése után, vagy a tétel sorszámának leütése után módosíthatjuk.

Az egyes tételek a következők:

0 Name

A mező neve, amely max. 15 karakter hosszú és megfelel a C nyelv szabályainak.

1 Description

A mező egysoros funkcióleírása, amely dokumentációs célokat szolgál.

2 Status line

A mező státusz információja, amely a mezőre való pozicionáláskor jelenik meg az ún. státusz sorban (ha van ilyen az aktuális lapon) státusz sor.

3 Type

A mező típusa, amely a következő kiválasztó menü alapján határozható meg (a menütételek mögött a típus magyarázata):

Byte	1-byte-os egész szám (char),
Int	2-byte-os egész szám,
Long	4-byte-os egész szám,

Float	4-byte-os lebegőpontos szám, (kb. 8 egész jegyre pontos)
dOuble	8-byte-os lebegőpontos szám,
Date	dátum, 4-byte-os,
Text	karaktertömb,
User	felhasználói (USER mező) típus,
Subform	al-form (SUBFORM mező) típus.

A típus (a szokásos módon) ráállás és ENTER, vagy a kiemelt betű leütésével választható ki.

4 Protected

Védettség, azaz a mező input vagy output mező-e. Lehetséges értékek

Yes	(output)
No	(input)

5 Formatted

Text típusú mezőhöz, ha annak nincs input függvénye, akkor beolvasási formátum rendelhető az "Assign format to the field?" kérdésre adott igenlő válasszal.

A formátum értelmezése a következő:

l,L	csak betű lehet (L esetén automatikus nagybetűre konvertálás),
a,A	betű vagy szám lehet (A esetén nagybetűre konvertálás),
n,N	nem lehet szóköz (N esetén nagybetűre konvertálás),
x,X	bármilyen karakter (X esetén nagybetűre konvertálás),
#	csak számjegy lehet,
9	számjegy vagy szóköz lehet.

Ha a formátum rövidebb a mező hosszánál, akkor a maradék helyek z formátum-elemnek minősülnek, tehát ezeken a pozíciókon bármilyen karakter megadható. Azokon a pozíciókon, ahol nem a fenti formátum-elemek szerepelnek, csak az adott karakter szerepelhet, futtatási fázisban ezekre a karakterekre nem is lehet ráállni.

5 Input function

A nem SUBFORM típusú mezőkhöz lehet mezőbeolvasó C függvényt rendelni, amely futtatási fázisban kiváltja a standard beolvasási módszert.

A USER típusú mezőkhöz az Input függvény hozzárendelése kötelező, hiszen csak a felhasználó tudhatja, hogy ez a típus hogyan kaphat értéket.

5 Load function

A SUBFORM típusú mezőknek kötelező megadni a betöltő függvényt. (Lásd az 5.2. Felhasználói függvények c. részt!)

6 Update function

A nem USER és SUBFORM típusú mezőkhöz egy Update (aktuálizáló és ellenőrző) függvény rendelhető, amely futtatási fázisban eldönti, hogy a mező új értéke alapján milyen tevékenységeket kell elvégezni (pl. más mezőkkel), ill. eldönti, hogy az érték helyes-e.

6 Screen function

A USER típusú mezőkhöz a Screen függvény hozzárendelése kötelező, hiszen csak a felhasználó tudhatja, hogy ez a típus milyen formában kerülhet ki a képernyőre.

6 Store function

A SUBFORM típusú mezőknek kötelező megadni a kitöltő függvényt. (Lásd az 5.2. Felhasználói C függvények c. részt!)

7 Has default

A nem SUBFORM típusú mezőkhöz default, kezdeti értéket lehet rendelni.

7 Modify subform

Ha a mező SUBFORM típusú, akkor ennek a funkciónak a kiválasztásával lehet az egy alrekord kezelését meghatározó al-form elemet definiálni, ill. módosítani. Ekkor a rendszer éppúgy engedi az al-form jellemzőket állítani, mint az elsődleges form-nál. Az al-form definiálása után az ESC billentyűvel lehet a módosítás kiválasztó menüjéhez visszatérni. (Lásd még a 4.12. SUBFORM mező definiálása című szakaszt!)

8 Length

A mező hossza a nem USER és a nem SUBFORM mezőknél. Állítása nagyon kényelmes, mivel egy számsor jelzi a mező aktuális hosszát és értelemszerűen a balra, jobbra nyilakkal lehet a hosszt csökkenteni, ill. növelni.

8 Lentgh

A USER és a SUBFORM mezőknél ez az adat a mezőhöz rendelt képernyőterület oszlopainak a száma. A mezőhöz rendelt terület mérete a kurzorvezérlő billentyűk segítségével értelemszerűen módosítható.

9 Signed

Egész típusoknál (BYTE, INT, LONG) azt adja meg, hogy az érték előjeles vagy sem. A lehetséges értékek: Yes (előjeles) és No (előjel nélküli).

9 Decimal places

Lebegőpontos típusoknál (FLOAT, DOUBLE) megadható, hogy a mező értéke hány tizedesjeggyel kerüljön kiírásra. Értelemszerűen a mező hosszának elegendőnek kell lennie a tizedesjegyek és a tizedespont számára.

9 Date format

A dátum (DATE) típusnál a dátum kiírási formája a következők valamelyike lehet (E: év, H: hónap, N: nap):

HH/NN/EE	(angol – English),
EE/HH/NN	(magyar – Hungarian),
NN/HH/EE	(német – German).

A megfelelő módot egy kiválasztási menü segítségével állíthatjuk be.

9 Depth of field

USER és SUBFORM mezőknél azt jelzi, hogy kiíraskor a mező hány sort foglal el a képernyőn. SUBFORM esetén, ha az al-form már definiált, akkor megjelenik az is, hogy az adott SUBFORM mezőben hány alrekord írható ki. Ez a 8. funkció (Length) segítségével módosítható.

4.12. SUBFORM mező létrehozása

Összetettsége miatt a SUBFORM típusú mező definiálásáról külön is szólnunk kell.

A SUBFORM típusú mező keretet ad az alrekordok kezelésére. Két különálló rész alkotja:

- maga a SUBFORM típusú mező, és
- az ezen belüli egy vagy több soros al-form, amely egy alrekordot kezel.

A mező definiálása tulajdonképpen három részből áll:

1. a SUBFORM mezőt körülvevő fő form-ba tartozó keretek, szövegek és esetleges I/O mezők definiálása,
2. a SUBFORM mező meghatározása,
3. az egy alrekordot kezelő (al)form definiálása.

A magyarázathoz egy egyszerű rekordleírás form-ját használjuk fel, amelyben megadható a rekord neve, kiírásra kerül a mezők száma és a rekord mezői definiálhatók (név, típus, hossz), módosíthatók és törölhetők.

Az első lépésben definiáljuk a form SUBFORM-ot körülvevő részét, a táblázatos formát elősegítő függőleges vonalazást és a fejléceket:

[illegible]

A második lépésben a SUBFORM mezőt a hozzá tartozó képernyőterülettel kell definiálni. A képernyőterület kijelölése a mező a SUBFORM típusúra való átállítása után végezhető el a kurzorvezérlő billentyűk, majd az ENTER használatával. Ezt a területet úgy kell meghatározni, hogy

- elegendő legyen egy vagy több alrekord megjelenítésére, és
- maradjon hely a SUBFORM mező és az al-form első oszlopa között, az esetlegesen nem látható alrekordok automatikus jelzésére.

Ha az automatikus jelzést nem akarjuk használni, az al-form a SUBFORM mezőhöz rendelt terület elejére húzható.

Most a kijelölt terület legyen a négy nyíl által behatárolt terület.

A terület definiálása után automatikusan létrejön egy al-form, amely egy soros, és a SUBFORM mezőhöz képest két pozícióval beljebb (jobbra) kezdődik. Azonnal lehetőséget kapunk az al-form definiálására is. Ha egy alrekord megjelenítésére nem elegendő egy sor, akkor ezen a ponton kell az al-form méretét megnövelni a már ismert Alt-W (Window) funkcióval.

A harmadik lépésben meghatározhatók az alrekord kezelését szolgáló al-form mezői. Ebben az esetben ez a Név, a Típus és a Méret mezők definiálását jelenti. Ha az al-form definiálása megtörtént, akkor ESC billentyűvel visszatérhetünk a SUBFORM mezőt módosító menühöz, ahol meg kell adnunk a Load (egy alrekordot az al-form-ba betöltő) és a Store (az al-formból egy alrekordot kimentő) C függvényt. Ezekre a FormEdit a mező nevét tartalmazó neveket ajánl fel. Mivel a FormEdit ekkor már ismeri a SUBFORM mező és az al-form méretének viszonyát, megjeleníti, hogy a SUBFORM mező hány alrekord megjelenítéséhez elegendő.

A SUBFORM mezőhöz tartozó al-form módosítása a későbbiekben a mező-módosító menü 7-es funkciójával kezdeményezhető.

4.13. Szövegkonstans létrehozása

Szövegkonstans beszúrását az aktuális pozícióra, az INS billentyű lenyomása után megjelenő, kiválasztási menü második tételének (Text) kiválasztásával kezdjük. Ezután beüthetjük a kívánt szöveget, ezt ENTER-rel érvényesítve.

A név megadásakor ékezetes betűket is definiálhatunk úgy, hogy a betű előtt megadjuk a három lehetséges ékezettípus (' : ") közül a megfelelőt (pl.: a "körülmény" szó megadása a "k:or:ulm'eny" karaktersorozattal lehetséges). Mivel a teljes magyar betűkészlet a legtöbb számítógépen nincs meg, ezért a rövid és a hosszú magánhangzók egy része egyformán jelenik meg.

A szöveg megadása után meghatározhatjuk a szöveg színét a következő kontroll billentyűk használatával:

- ^F a kiírási (Foreground) szín változtatása,
- ^B a háttérszín (Background) változtatása,
- ^S a szín változtatása a form-hoz rendelt öt elemű színpalettából (lásd még a futtatórendszer `Form_Colors()` függvényét).

4.14. Szövegkonstans törlése

A szövegkonstans a rá való pozicionálással, majd a DEL billentyű lenyomásával törölhető. A törlési szándékot meg kell erősíteni az "Are You sure to delete text?" kérdésre adott igenlő válasszal.

4.15. Szövegkonstans módosítása

Az aktuális szövegkonstans módosítása a szövegre való ráállás után az INS billentyűvel kezdeményezhető. Maga a módosítás a beszúrással megegyező módon végezhető.

4.16. Státusz sor beállítása

Ha a mezőknél az úgynevezett státusz információ szövegét megadtuk, akkor a mezőre való pozicionáláskor ez a szöveg megjelenik a form aktuális oldalán lévő státusz sorban, mind a FormEdit használatakor, mind pedig a futtatási fázisban.

A státusz sor definiálása nem más mint egy szövegkonstans definiálása, amely:

- minimum 10 karakter hosszú, és
- csupa pont (.) karaktereket tartalmaz.

Ha a form több lapos, akkor célszerű minden lapra, kb. ugyanarra a képernyőrészre definiálni a státusz sort.

4.17. Grafikus elem létrehozása

Grafikus elem beszúrását az aktuális pozícióra az INS billentyű lenyomása után megjelenő kiválasztási menü harmadik tételének ("Box, row, column") kiválasztásával kezdjük.

Ezután

- a kurzorvezérlő billentyűk segítségével beállíthatjuk a négyszög vagy az oszlop vagy a vonal méretét,
- a < és a > (vagy , és .) billentyűkkel beállíthatjuk a vonal típusát (dupla, szimpla, vízszintesen dupla - függőlegesen szimpla és vízszintesen szimpla - függőlegesen dupla),
- a ^F (Foreground - kiírási szín), ^B (Background — háttérszín) és ^S (szín beállítása a form öt elemű — Standard — színpalettájából, lásd még a futtatórendszer **Form_Colors()** függvényét) billentyűkkel beállíthatjuk a grafikus elem színét.

4.18. Grafikus elem törlése

A grafikus elem az ábra bal felső sarkára való pozicionálással, majd a DEL billentyű lenyomásával törölhető. A törlési szándékot meg kell erősíteni az "Are You sure to delete graphic item?" kérdésre adott igenlő válasszal.

4.19. Grafikus elem módosítása

Az aktuális grafikus elem módosítását, az elem bal felső sarkára pozicionálás után, az INS billentyűvel kérhetjük. Maga a módosítás a beszúrással megegyező módon végezhető.

4.20. Mozgatás

Az aktuális mező vagy szövegkonstans mozgatását az Alt-M (Move) billentyű lenyomása után két lépésben végezhetjük el.

- Először a kurzorvezérlő billentyűk segítségével ki kell jelölni, és az ENTER-rel ezt rögzíteni kell a mozgatandó mezők és szövegkonstansok csoportját. A kijelölt terület inverz színnel jelenik meg. Ha a mozgatás kezdetén egy form-elem első pozícióján áll a kurzor, akkor a kijelölt terület az adott form-elem területe lesz.
- Ezután, szintén a kurzorvezérlő billentyűk segítségével, az átszínezéssel kijelölt terület (tartalma) a kívánt helyre mozgatható. A mozgatást az ENTER billentyű segítségével fejezhetjük be.

Ha a mozgatás eredményeként egymást lefedő mezők és szövegkonstansok jönnek létre, akkor a mozgatást a FormEdit visszautasítja. Ha mozgatás eredményeként egy grafikus elem a form-hoz rendelt képernyőterületről kilóg, akkor a grafikus elem mérete a szükségesre igazítódik.

4.21. Mezők és szövegkonstansok másolása

Az aktuális mező vagy szövegkonstans másolása az Alt-Y (copY) billentyűvel lehetséges. A billentyű lenyomása után az aktuális form-elem inverz színezésűre vált. Az átszínezett képernyőrész a kurzor-billentyűk segítségével elmozgatható egy tetszőleges (más mező vagy szövegkonstans által nem takart) képernyőrészre. A másolást az ENTER billentyűvel érvényesíthetjük. Ekkor

- mező esetén meg kell adni az átmásolt mező nevét (hiszen a form-on belül nem lehet két azonos nevű mező),
- szövegkonstans esetén a szövegkonstans módosításához hasonlóan beállíthatjuk az átmásolt szöveg színét.

4.22. A mezők áttekintő listája

A képernyőn listázhatjuk a már definiált mezők jellemzőit az Alt-L (List) billentyű lenyomásával. A megjelenő listán az aktuális form, ill. al-form információi is szerepelnek.

4.23. Az aktuális mező jellemzői

Az aktuális mező jellemzőinek részletes listázása a "?" billentyű leütésével érhető el.

4.24. Billentyűk elrendelése

Az M-FORM lehetőséget ad bármely billentyű elrendelésére úgy, hogy futtatási fázisban a billentyű leütésekor egy felhasználói C függvény hívódik meg és a leütött billentyű ennek visszatérési értéke lesz.

A billentyű elrendelése az Alt-K (**K**ey assignment) billentyű leütésével kezdhető, ekkor megjelenik a már elrendelt billentyűk neve és C függvénye. Az INS billentyű leütésével kérhetjük az új billentyűelrendelést. Ekkor a "Press the key to assign" kiírás után le kell ütni az elrendelendő billentyűt, majd az "Enter the name of C function" kiírás után meg kell adni a felhasználói C függvény nevét. Az elrendelt függvény megadása ESC billentyűvel leállítható. Egy létező elrendelés a DEL billentyűvel törölhető. Egy C függvény neve az ENTER billentyű leütése után módosítható. Az első billentyűelrendelés kérésekor a rendszer az INS billentyű leütésével megegyező módon működik.

4.25. Aktuális képernyőpozíció lekérdezése

Az Alt-P (**P**osition) billentyű hatására a pozíció kiírási funkció kapcsolható be- ill. ki. Bekapcsolt állapotban a képernyőn minden helyváltoztatás után kiírásra kerül az aktuális sor (abszolút és a lap tetejéhez viszonyított), sorszáma, valamint az oszlop- és a lapszám.

4.26. A képernyő újraíratása

A teljes képernyő újraíratása az Alt-R (Redisplay) billentyűvel kérhető. Erre akkor lehet szükség, ha a képernyő mozgatása, méretmódosítása, vagy a SUBFORM mező átírása után bizonyos korábban kiírt részek nem törlődnek automatikusan a képernyőről.

4.27. Kimentés file-ba

A definiálás alatt lévő form aktuális állapotát kimenthetjük file-ba az Alt-S (Save) billentyű leütésével.

A rendszer mentés előtt megkérdezi az output file nevét. Az .FTX típust nem kell megadni, mivel a file-típust a rendszer mindenképpen erre az értékre állítja.

Ha a form-file már létezik, a felülírási szándékot az "Overwrite existing file?" kérdésre adott igenlő válasszal meg kell erősíteni.

4.28. Programrészek generálása

A FormEdit különböző C programrészek generálásával nagymértékben könnyíti a programozói munkát.

A generált forrásnyelvi file-ok a következők:

1. Header file

az egyes mezők C programból való azonosítására (.FH).

2. Felhasználói függvények táblája

a felhasználói C függvények bekapcsolása a futtatórendszerbe (.FCT).

3. Felhasználói függvények prototípusa

- a felhasználói C függvények definíciója és paraméterlistája (.FCC).
4. A form-ot dinamikusan létrehozó forrás
form-file nélküli alkalmazói program létrehozásához (.FIC).

4.28.1. Header-file

A futtatórendszer komplexebb használatához az egyes mezők sorszámát a programozónak ismernie kell. Ezenkívül hasznos lehet, ha a felhasználónak van egy olyan struktúra definíciója, amely megegyezik a futtatórendszer belső bufferének beosztásával.

A generált header-file ezt a két fajta definíciót tartalmazza:

A mezők sorszáma: F_ és a mező neve.

A struktúra: S és a form neve.

Ezt a definíciós header-file-t az Alt-H (generate Header) billentyű leütésével lehet kérni. A file neve a form neve és az .FH típus lesz. Ha a FormEdit hívásakor megadtuk a *headerpath* paramétert, akkor a file ebbe a könyvtárba íródik.

Figyelem!

A futtatórendszer a form belső bufferét maximális tömörítettséggel kezeli. Ezért, ha a generált header-file struktúra-definícióját használjuk, akkor a fordításkor a Microsoft compiler /Zp opcióját kell használni!

4.28.2. Felhasználói függvények táblája

A futtatórendszer és a felhasználói programban leírt függvények kapcsolatát a felhasználói függvények táblájának segítségével lehet megteremteni. A

generált tábla párba állítja a felhasználói függvények nevét és a tényleges függvényt.

A file generálását az Alt-F (generate Function table) billentyűvel kérhetjük. A file neve, a form neve és a .FCT típus lesz. Ha a FormEdit hívásakor megadtuk a *headerpath* paramétert, akkor a file ebbe a könyvtárba íródik.

4.28.3. Felhasználói függvények prototípusa

A felhasználói függvények megírásában sokat segít a függvények prototípusának automatikus generálása. Ezt a funkciót az Alt-G (generate C functions) billentyűvel kérhetjük. Hatására egy C forrás-file képződik, amely tartalmazza a form és az esetleges al-formok összes felhasználói C függvényének keretét, paraméterlistáját magyarázattal ellátva, valamint néhány tipikusnak mondható utasítást, amely alapján a C függvény már könnyen megírható.

A file neve, a form neve és a .FCC típus lesz. Ha a FormEdit hívásakor megadtuk a *headerpath* paramétert, akkor a file ebbe a könyvtárba íródik.

4.28.4. A form-ot dinamikusan létrehozó forrás

Ha a felhasználó úgy dönt, hogy form-ja végleges, és a form-ot használó programot úgy akarja használni, hogy a form-file-t (.FTX) ne kelljen terjesztetni, akkor lehetőség van egy olyan C forrás-file generálására, amely dinamikusan hozza létre az egyébként form-file-ban leírt form-okat. E forrás beszúrásával, és benne deklarált C függvény hívásával kikerülhető a beolvasás a form file-ból.

A generálást az Alt-I (generate Initializing source) funkcióval kérhetjük.

Az eredmény egy .FIC típusú file lesz, amely tartalmazza az egyes form-okhoz tartozó billentyűkhöz rendelt C függvények tábláját és egy C függvényt, amely a **Form_Crea()**, **Box_Crea()** stb. függvények segítségével létrehozza a megfelelő form-okat, és visszaadja az első form sorszámát.

A C függvény neve az első form neve és a **_crea** string lesz. Pl.: ha a fő form neve **SAMPLE**, akkor a rutin neve **SAMPLE_crea()** lesz.

A függvénynek paramétere nincs. Visszatérési értéke -1, ha a form-ok létrehozása sikertelen volt, ill. az első (fő) form sorszáma.

Ha a FormEdit hívásakor megadtuk a *headerpath* paramétert, akkor a file ebben a könyvtárban képződik.

4.29. Kilépés az operációs rendszerbe

A DOS operációs rendszerbe az F9 billentyűvel léphetünk ki. Itt normál DOS módban dolgozhatunk, majd az EXIT parancs beütésével visszatérhetünk a FormEdit rendszerébe és folytathatjuk a form definiálását.

4.30. A parancsok összefoglalása

Az F1 billentyűvel a parancsokról a következő rövid angol nyelvű összefoglalás kérhető a képernyőre:

Position : ^ V < > Space pozícionálás

Position : Home End PgUp PgDn

info about actual field :	?	Az aktuális mező jellemzői
quit :	ESC	Kiszállás
Insert/Modify field/text/box :	INS	Beszúrás/módosítás
Delete field/text/box :	DEL	Törlés
change Color :	Alt_C	Sínbeállítás
Description of form :	Alt_D	Form funkcióleírás módosítása
generate C Function table :	Alt_F	C függvénytábla generálása
Generate C function source :	Alt_G	C függvény forrás generálása
generate Header file :	Alt_H	Headerfile generálása
generate Initializing source :	Alt_I	Dinamikus, form-létrehozó függvény generálása
define/modify Key function :	Alt_K	Billentyű elrendelés
List fields :	Alt_L	Mezők leírásának megjelenítése
Move selected elements :	Alt_M	Form-elemek mozgatása
display Position :	Alt_P	Pozíció folyamatos kiírása

Redisplay form :	Alt_R	A képernyő újraírása
Save form :	Alt_S	A form kimentése
copY field or text :	Alt_Y	Mező vagy szöveg másolása
change Window :	Alt_W	A form ablakának módosítása
System mode (DOS window) :F9		Kilépés az operációs rendszerbe

5. fejezet

A programozás

A programozó az M-FORM által biztosított adatbeviteli lehetőségeket viszonylag egyszerűen tudja beépíteni programjába.

Tennivalói:

- A FormEdit segítségével az adatbevitel formájának, tartalmának és a beépített felhasználói függvények csatlakozási pontjainak meghatározása.
- A futtatórendszer és a saját modulok köré keretprogram készítése.
- A csatlakozási pontokban hívott felhasználói függvények megírása.
- A modulok fordítása és szerkesztése.
- Az adatbeviteli program futtatása.

Valójában programozási feladatot csak a második és a harmadik lépés jelent csak. Egyszerű beviteli programoknál ezek a lépések standard megoldásokra vezethetők vissza, tehát lehet, hogy a programozónak semmit nem kell programoznia. A technikai jellegű lépések (a fordítás, a szerkesztés) automatizálhatók, batch eljárás vagy a make program segítségével.

5.1. A futtatórendszer függvényei

A felhasználó a futtatórendszer rutinjain keresztül használhatja a FormEdit-t létrehozott form-ot. A rutinok típusa és paramétereik típusa a FORM.H header-file-ban vannak definiálva szintaktikai ellenőrzés céljából. Ezért azokban a C modulokban, ahol a felhasználó hivatkozik valamelyik, a futtatórendszerhez tartozó rutinra, érdemes a FORM.H file-t **include**-olni. Így a

Microsoft C fordítóval (/Zs/DLINT_ARGS opciók) kiszűrhetőek a függvény hívásának egyszerű formai hibái.

A FORM.LIB függvényeit funkciójuk szerint a következő csoportokra oszthatjuk:

- a form megnyitása
Form_Read(), Form_Crea(), Field_Crea(), Text_Crea(), Box_Crea();
- az aktív form állítása
Form_Select(), Form_Close(), Search_Form(), Form_Do(), FormF_Do();
- a felhasználói C függvények illesztése a futtatórendszerhez
Set_Form_Funcs(), Del_Form_Func();
- a form információinak lekérdezése
Form_Buffer(), Form_Size(), Field_Number();
- a képernyőn való megjelenítés
Field_Put(), Form_Clear(), Form_Colors(), Form_Fields(), Form_Text(), Form_Wsave(), Form_Wclose(), Form_First_Text(), Form_Last_Text();
- a form ciklikus (rekordonkénti) kezelése
Form_Init(), Form_Exec(), Form_Break(), Form_Changed(), Field_Select(), Field_Def_All(), Field_No_Valued();
- a mezők értékének kezelése
Field_Set(), Field_F_Set(), Field_Get(), Field_F_Get(), Field_Edit();
- a mező jellemzőinek lekérdezése
Field_Act(), Field_Col(), Field_Dec(), Field_Is_Carry(), Field_Is_Prot(), Field_Length(), Field_List(), Field_Name(), Field_Offset(), Field_Row(), Field_Sernum(), Field_Size(), Field_Type(), Field_Valued();
- a mező jellemzőinek beállítása
Field_Carry(), Field_Def(), Field_Protect(), SubForm_Set();
- a standard mezőbeolvasás
Edit_Fmt(), Edit_Num();
- a formátumos mezők kezelése

Check_Fmt(), **Compress_Fmt()**, **Copy_Fmt()**,
Extract_Fmt(), **Fill_Fmt()**;

- dátumkezelés
atodat(), **convdat()**, **sysdate()**;
- konverziós rutinok
atolu(), **atou()**, **strtotxt()**, **txttostr()**,
Key_Name(), **Key_Scan()**.

Az egyes függvények leírásánál gyakran találkozhatunk azzal a kijelentéssel, hogy hiba esetén a függvény (hibaüzenet kiírása után) abortáltatja az alkalmazói programot. Ennek az a magyarázata, hogy ezek programozási hibák, tehát az alkalmazói program futtatása értelmetlen. Így az abortálás a helyes alkalmazói keretprogram megírására kényszerít.

5.1.1. Form megnyitása

A programozó számára egy form két módon nyitható meg:

1. A **Read_Form()** függvény meghívásával,
2. A form dinamikus létrehozásával.

Az első módszer a FormEdit outputjaként keletkezett form-file-ból olvassa be az információkat, így a form-ban a programozást nem érintő változtatások (pl. szín változtatás, beolvasási formátum, default érték) esetén a programot nem kell újrafordítani, cserébe az alkalmazói programmal együtt kell adni a form-file-t.

A második módszer esetén a **Form_Crea()**, **Box_Crea()**, **Text_Crea()** és **Field_Crea()** függvények hívásával nyithatjuk meg a form-ot. Ebben az esetben bármely változtatás a form-ban a program újrafordítását igényli. Ezt a módszert segíti a FormEdit azon generálási funkciója, amely a form-ot dinamikus létrehozó forrásmodult generálja.

Form_Read

Funkció: A form-file-ban tárolt form-leírások beolvasása.

Használat:

```
#include <FORM.H>
int Form_Read(fname)
char *fname; a form-file neve
```


- Leírás:** A rutin bolvassa a megadott file-ból az abban lévő form-leírásokat, amelyeket egyben meg is nyit. Az aktuális form a file-ban lévő első form lesz.
- A rutin használata kikerülhető, ha a FormEdit-tel a formot dinamikusan létrehozó C függvényt generáltunk, és ennek hívása után a **Form_Select()** függvénnyel aktivizáljuk a létrehozott formot.
- Visszatérés:** -1 hiba esetén, ill. a hívás után aktív form (a file-ban lévő első form) azonosító sorszáma.
- Lásd még:** **Form_Crea()**, **Form_Select()**

Form_Crea

Funkció: Form dinamikus létrehozása.

Használat:

```
#include <FORM.H>
#include <SETATTR.H>
#include <KBGET.H>
int Form_Crea(n,d,ur,lc,lr,rc,b,i,e,m,h,ak)
char      *n;           a form neve
char      *d;           a form funkcióleírása
int        ur;          felső sor
int        lc;          bal oszlop
int        lr;          alsó sor
int        rc;          jobb oszlop
int        b;           alapszín
int        i;           input szín
int        e;           editálási szín
int        m;           üzenet szín
int        h;           help szín
struct ASSKEY *ak;      hozzárendelések
```

Leírás: A rutin segítségével programból lehet létrehozni dinamikusan egy formot. A színek a <SETATTR.H> header-file alapján adhatók meg. Az *ak* paraméter a <FORM.H> header-file-ban definiált ASSKEY struktúrával megadható táblázat címe, amely az egyes billentyűkhöz rendelt felhasználói C függvényeket tartalmazza.

Visszatérés: -1 hiba esetén, ill. a létrehozott form azonosító sorszáma.

Lásd még: **Form_Select()**, **Form_Read()**

Példa:

```

#include <STATTR.H>
#include <KBGET.H>
#include <FORM.H>

struct ASSKEY ak[] = {
    { CTR_PGUP ,"Scroll_Up"      },
    { CTR_PGDN ,"Scroll_Dn"      },
    { -1       , 0               }
};

main()
{
    int      form;
    form = Form_Crea("SAMPLE",
                    "Sample form only",
                    0,0,24,79,
                    GRAY,
                    WHITE,
                    COLOR(BLACK,GRAY),
                    COLOR(BLACK,RED),
                    COLOR(BLACK,GREEN),
                    ak);
}

```

Box_Crea

Funkció: Grafikus elem dinamikus létrehozása.

Használat:

```

#include <FORM.H>
#include <FRAME.H>
#include <SETATTR.H>
int Box_Crea(ur,lc,lr,rc,ft,clr)
int ur;   felső sor

```

```
int lc;    bal oszlop
int lr;    alsó sor
int rc;    jobb oszlop
int ft;    keret típus
int clr;   szín
```

Leírás: A rutin segítségével dinamikusan hozhatunk létre tetszőleges vízszintes vagy függőleges vonalat, ill. keretet az aktív form-ban.

Visszatérés: -1 hiba esetén, egyébként a grafikus elem sorszáma.

Példa: A képernyő bekeretezése:

```
#include <FRAME.H>
#include <SETATTR.H>
.
.
Box_Crea(0,0,24,79,HSV5,GRAY);
```

Text_Crea

Funkció: Szövegkonstans dinamikusan létrehozása.

Használat:

```
#include <FORM.H>
#include <SETATTR.H>
int Text_Crea(txt,l,c,clr)
char    *txt;    a szöveg
int      l;      sor
int      c;      oszlop
int      clr;    szín
```

Leírás: A rutinnal dinamikusan beszúrhatunk az aktuális form-ba egy új szövegkonstanst.

Visszatérés: -1 hiba esetén, egyébként a szövegkonstans sorszáma.

Lásd még: `Form_First_Text()`,
`Form_Last_Text()`

Példa:

```
#include <FORM.#>
#include <SETATTR.H>
Text_Crea("Személyi nyilvántartás",0,16,GRAY);
```


Field_Crea

Funkció: Mező dinamikus létrehozása.

Használat:

```
#include <FORM.H>
int Field_Crea(n,d,s,t,p,f,r,c,p1,p2,if,uf,def)
char    *n;      a mező neve
char    *d;      a mező leírása
char    *s;      státus sor
int      t;      típus
int      p;      védett (output) mező (0,1)
int      f;      van formátum? (0,1)
int      r;      sor
int      c;      oszlop
int      p1;     lásd lent
int      p2;     lásd lent
char    *if;     input C függvény/format
char    *uf;     update C függvény
char    *def;    default érték
```

Leírás: A rutinnal az aktuális form-ba szúrhatunk be egy új elemet dinamikusán.

A típus a <FORM.H> header-file-ban felsorolt értékek valamelyike lehet (FLD_BYTE...FLD_SUBF).

A *p1* paraméter

USER és SUBFORM esetén a mező területének alsó sora, egyébként a mező hossza.

A *p2* paraméter jelentése a típustól függően:

BYTE, INT és LONG esetén; ha 0, akkor a mező előjeles egész, egyébként előjel nélküli,

FLOAT és DOUBLE esetén: a decimális számjegyek száma,

DATE esetén: dátum forma (0: angol, 1: magyar, 2: német),

USER és SUBFORM esetén: a mező jobb oszlopa.

Az *if* paraméter

SUBFORM típus esetén a betöltő rutin neve,

USER esetén az input függvény neve,

a többi típusnál: ha az *f* paraméter nem 0, akkor a formátum; az input függvény neve egyébként (opcionális).

Az *uf* paraméter

SUBFORM típus esetén a kiíró rutin neve,

USER esetén a képernyőn megjelenítő függvény neve, a többi típusnál az update (ellenőrző) függvény neve (opcionális).

A *def* paraméter

a típusnak megfelelő default értéket tartalmazó string vagy 0.

Visszatérés: -1 hiba esetén, egyébként a mező sorszáma.

Példa: Egy nevet tartalmazó TEXT típusú mező létrehozása

```
Field_Crea("név",
    "A dolgozó neve",
    "Adja meg a dolgozó nevét!",
    FLD_TEXT,
    0,          /* input mező */
    1,          /* van formátum */
    2,18,       /* pozíció */
    32,         /* hossz */
    0,          /* TEXT-nél nincs jelentése */
    "L",        /* a formátum */
    NULL,       /* nincs ellenőrző függvény */
    NULL);     /* nincs default érték */
```

5.0.2. Aktív form állítása

Mivel az M-FORM egyszerre több megnyitott form meglétét is lehetővé teszi, szükség van az egyes form-ok közötti váltásra. Aktív form-nak az utoljára kiválasztott form-ot nevezzük. Egy form a következő módokon választható ki:

- Explicit módon a **Form_Select()** függvény hívásával.
- A **Form_Read()** függvény hívásakor az aktuális form a form-file-ban lévő első form lesz.

- A **Form_Exec()** végrehajtása során, ha **SUBFORM** mezőre lépünk, akkor az aktuális form a **SUBFORM** mezőhöz tartozó al-form lesz (a **SUBFORM** mező elhagyásakor az aktuális form ismét a **SUBFORM** mezőt magába foglaló form lesz).
- A **Form_Close()** hívása után nincs aktuális form.

Form_Select

Funkció: Form kiválasztása.

Használat:

```
#include <FORM.H>
int Form_Select(sernum)
int sernum; a kívánt form sorszáma
```

Leírás: A már megnyitott, de éppen nem aktív form kiválasztása. Használata csak több form párhuzamos, váltakozó alkalmazásánál értelmes.

Visszatérés: -1, ha a megadott sorszámu form nincs megnyitva, egyébként a kiválasztott form sorszáma.

Lásd még: **Search_Form()**

Form_Close

Funkció: Az aktív form lezárása és az általa elfoglalt memóriaterület felszabadítása

Használat:

```
#include <FORM.H>
Form_Close()
```

Leírás: A **Form_Read()** vagy a **Form_Crea()** rutinnal létrehozott form által lefoglalt memóriaterületet szabadítja fel. A **Set_Form_Funcs()** által beállított C függvénytábla változatlan marad, mivel lehetséges, hogy egy-egy felhasználói C függvényt több form is használ (pl.: billentyűk elrendelése).

Visszatérés: Nincs.

Lásd még: **Del_Form_Func()**

Search_Form

Funkció: Megkeresi a megnyitott form-ok között a megadott nevűt.

Használat:

```
#include <FORM.H>
int Search_Form(formname)
char    *formname;    a form neve
```

Leírás: A már megnyitott form-ok között megkeresi a megadott nevűt és visszaadja annak sorszámát.

Visszatérés: -1, ha nincs ilyen nevű megnyitott form, egyébként a form sorszáma.

Lásd még: Form_Select()

Form_Do

Funkció: Mezőkezelő függvény végrehajtása nem az aktív form-ban.

Használat:

```
#include <FORM.H>
int Form_Do(formsn,fp,fieldnum,arg)
int    formsn;        a form sorszáma
int    (*fp)();       végrehajtandó függvény
int    fieldnum;      a kívánt mező sorszáma
char    *arg;         átadandó paraméter
```

Leírás: E rutinnal egy mezőkezelő függvényt lehet végrehajtani a nem aktív, hanem a megadott sorszámú form-ban. Tipikus esete ennek, amikor a képernyőn egyszerre több form segítségével kezeljük az adatokat és az aktuális formhoz tartozó mező értékének megváltozása hatással van egy, a másik formhoz tartozó mező értékére. A leggyakrabban így használt mezőkezelő függvények:

Field_Set(), Field_Put(), Field_Get()

Visszatérés: A mezőkezelő függvény visszatérési értéke.

Lásd még: FormF_Do()

Példa: Az aktuális *menny* mező ellenőrző függvénye felfrissíti a másik (2-es sorszámú) form *ertek* mezőjét a szintén a másik formhoz tartozó *ar* mező alapján:

```
int menny_update(fieldnum,addr)
```



```

/* Update & check function of field 'menny' */
int      fieldnum;    /* serial num. of the field */
float    *addr;       /* address in the form buffer*/
{
    float  ar;
    float  ertek;
    Form_Do(2, Field_Get, F_ar, (char *)&ar);
    ertek = *addr * ar;
    Form_Do(2, Field_Set, F_ertek, (char *)&ertek);
    Form_Do(2, Field_Put, F_ertek, NULL);
    Field_Def(fieldnum, 1);    /* sign that ok. */
    return(F_UPD_OK); /* = GOOD VALUE */
}

```

FormF_Do

Funkció: Mezőkezelő függvény végrehajtása al-formból.

Használat:

```
#include <FORM.H>
```

```
int FormF_Do(fp, fieldnum, arg)
```

```
int      (*fp)();          végrehajtandó függvény
```

```
int      fieldnum;         a kívánt mező sorszáma
```

```
char     *arg;             átadandó paraméter
```

Leírás: E rutinnal egy mezőkezelő függvényt lehet végrehajtani az al-formból a hívó form egy mezőjére. Tipikus esete ennek, amikor egy alrekord változtatása kihatással van a rekord információira (pl.: összeg mező). A leggyakrabban így használt mezőkezelő függvények:

Field_Set(), Field_Put(), Field_Get()

Visszatérés: A mezőkezelő függvény visszatérési értéke.

Lásd még: Form_Do()

5.1.3. Felhasználói függvények illesztése

A form számára megadott felhasználói függvények és a futtatórendszer közötti kapcsolat egy táblázat alapján teremthető meg. Ezt a táblát a `FormEdit` a form-file alapján generálja le. Egy form megnyitása után a hozzá tartozó táblát a futtatórendszer számára ismertté kell tenni. Erre szolgál a `Set_Form_Funcs()` függvény. Ez a futtatórendszer belső táblájába beilleszti az aktuális táblából azokat a függvény-hivatkozásokat, amelyek még nem szerepeltek (hiszen lehetséges, hogy ugyanarra a felhasználói függvényre több form-ban is van hivatkozás).

A `Del_Form_Func()` függvény segítségével a már a `Form_Close()` függvénnyel lezárt form — más form-ban nem hivatkozott — függvényeit törölhetjük a futtatórendszer belső táblájából. Erre általában csak nagyon szűk memóriakapacitás esetén, és sok form használatakor van szükség.

Set_Form_Funcs

Funkció: A felhasználói C függvények táblájának átadása a futtatórendszer számára.

Használat:

```
#include <FORM.H>
int Set_Form_Funcs( cft)
struct CFUNC *cft; a FormEdit által generált
                  C függvénytábla
```

Leírás: Ennek a függvénynek segítségével lehet a form-hoz rendelt felhasználói függvények tábláját a futtatórendszer számára ismertté tenni. E függvény hívása után a futtatórendszer a form által hivatkozott felhasználói függvényeket el tudja érni, a hivatkozásokat futtatási időben fel tudja oldani.

Visszatérés: A form-rendszer által hivatkozott felhasználói C függvények száma.

Lásd még: `Del_Form_Func()`

Példa: A "SAMPLE" form beolvasása, majd a függvényhivatkozások feloldása:

```
#include <FORM.H>
#include "sample.fct"
```



```
main()
{
    if(Form_Read("SAMPLE") < 0)
        exit(4);
    Set_Form_Funcs(SAMPLE_FCT);
}
```

Del_Form_Func

Funkció: Felhasználói C függvény törlése a futtatórendszer belső táblázatából.

Használat:

```
#include <FORM.H>
int Del_Form_Func(name)
char    *name;           a törlendő C függvény neve
```

Lefrás: A lezárt form-ok által már nem használt felhasználói C függvényt törli a futtatórendszer belső táblázatából.

Visszatérés: -1, ha a függvény nincs a belső táblában, ill. a törölt függvény táblabeli sorszáma.

Lásd még: Set_Form_Funcs(), Form_Close()

5.1.4. Form információk lekérdezése

Általánosított programrészek megírásához nélkülözhetetlen, hogy az aktuális programrészben a form információi dinamikusan lekérdezhetők legyenek. A következő rutinok elsősorban a futtatórendszer belső bufferének dinamikus elérésére használatosak.

Form_Buffer

Funkció: A form belső buffercímének visszaadása.

Használat:

```
#include <FORM.H>
```

```
char *Form_Buffer()
```

Leírás:

Az aktuális form belső bufferének címét adja vissza. Elsősorban a direkt betöltés esetén van szerepe. A form bufferének használata nagy körütekintést igényel, mivel a bufferben a mezők értékei, a mezők sorrendjének megfelelően, belső ábrázolással vannak tárolva. Használata akkor hasznos, ha a form és a külső (felhasználói) buffer szerkezete pontosan megegyezik. Ekkor a **Form_Size()** függvénnyel visszaadott értéket (a buffer hosszát) felhasználva egy memóriamásolással áttöltethető az egyik buffer tartalma a másikba.

Figyelmeztetés!

Ha a visszkapott címet mint a *formnév.H* header-file-ban definiált struktúrát használjuk, akkor a Microsoft C fordítót a -Zp opcióval kell indítani, mert a fordító alapértelmezésben szóhatárra igazítja a struktúraelemeket, a FORM.LIB rutinjai pedig tömörítve használják.

Visszatérés: Az aktuális form belső bufferének címe.

Lásd még: **Form_Size()**

Form_Size

Funkció: A form belső bufferméretének visszaadása.

Használat:

```
#include <FORM.H>
```

```
int Form_Size()
```

Leírás:

Visszaadja az aktuális form belső bufferének a méretét. Elsősorban a direkt betöltés esetén van szerepe.

Visszatérés: A belső buffer mérete (megegyezik a generált .H header-file-ban definiált struktúra méretével).

Lásd még: **Form_Buffer()**

Field_Number

Funkció: A form mezőszámának visszaadása.

Használat:

```
#include <FORM.H>
```

```
int Field_Number()
```

Leírás: Visszaadja, hogy az aktuális form-ban hány mező van. Az érték elsősorban a form összes mezőjének ciklusban való kezelésére alkalmas.

Visszatérés: A form mezőinek száma.

Példa: Az összes mező "megmaradó értékű" jellemzőjének beállítása:

```
#include <FORM.H>
```

```
int      n,i;
```

```
n = Field_Number();
```

```
for(i = 0; i < n; ++i)
```

```
    Field_Carry(i, 1);
```

5.1.5. Megjelenítés képernyőn

A futtatórendszer funkciói közül a képernyőn való megjelenítés kiemelt jelentőségű, hiszen az operátor a képernyőről kapja az összes információt a kezeléshez. Ezért ezek a funkciók is változatos szinten állnak a programozó rendelkezésére.

Field_Put

Funkció: Egy mező kiírása a képernyőre.

Használat:

```
#include <FORM.H>
```

```
Field_Put(fieldnum)
```

int *fieldnum*; a kiírandó mező sorszáma

Leírás:

Az aktuális form megadott mezőjét írja ki a képernyő megfelelő helyére. Ha a mező még nem kapott értéket, akkor a mező területén egy aláhúzás karakterekből álló vonal jelenik meg. USER mező esetén a **Field_Put()** a mezőhöz rendelt kiíró függvényt hívja meg, megfelelő paraméterezéssel. SUB-FORM mező esetén a rutin az alrekordok aktuális számától függően meghívja a betöltő függvényt, és az alrekordokat ennek megfelelően íratja ki a képernyőre.

Visszatérés: Nincs.

Lásd még: **Form_Fields()**, **Sub_Form_Set()**

Form_Clear

Funkció: A form-hoz tartozó képernyőrész törlése.

Használat:

```
#include <FORM.H>
```

```
Form_Clear()
```

Leírás:

Törli az aktuális form-hoz tartozó képernyőrészt a form-hoz rendelt alapszínnel.

Visszatérés: Nincs.

Form_Colors

Funkció: A form-ban meghatározott színek lekérdezése, ill. átállítása.

Használat:

```
#include <FORM.H>
```

```
Form_Colors(func, colors)
```

int *func*; lekérdezés/beállítás

char *colors*[5]; a színek tömbje

Leírás: A rutin az első paraméter értékétől függően működik:
Ha a *func* paraméter értéke 0, akkor a form-hoz rendelt öt színt az átadott *colors* tömbbe tölti.
Ha a *func* paraméter nem 0, akkor a *colors* tömbben lévő öt színt rendeli a form-hoz. Ha a form-hoz rendelt szövegkonstansok és grafikus elemek színe megegyezik valamelyik eredeti színnel, akkor e form-elemek színe is az új színre változik.

Visszatérés: Nincs.

Form_Text

Funkció: A form grafikus elemeinek és szövegkonstansainak kiírása a képernyőre.

Használat:

```
#include <FORM.H>
```

```
Form_Text()
```

Leírás: A rutin a form (aktuális lapjának) grafikus és szövegkonstans elemeit írja ki a képernyőre.

Visszatérés: Nincs.

Form_Fields

Funkció: A form mezőinek kiírása a képernyőre.

Használat:

```
#include <FORM.H>
```

```
Form_Fields()
```

Leírás: A rutin a form (aktuális lapjának) input és output mezőit írja ki a képernyőre.

Visszatérés: Nincs.

Lásd még: `Field_Put()`

Form_Wsave

Funkció: A megnyitott form által kezelt képernyőterület mentése.

Használat:

```
#include <FORM.H>
```

```
int Form_Wsave()
```

- Leírás:** A rutin az aktív form képernyőterületét menti úgy, hogy a képernyő tartalma a **Form_Wclose()** függvénnyel visszaállítható. A rutint a **Form_Clear()** - **Form_Text()** - **Form_Fields()** függvények hívása előtt célszerű meghívni.
- Visszatérés:** -1, ha nincs elegendő memória a képernyő mentéséhez, egyébként 0.
- Lásd még:** **Form_Wclose()**

Form_Wclose

- Funkció:** A **Form_Wsave()** függvénnyel mentett képernyőtartalom visszaírása.
- Használat:**
- ```
#include <FORM.H>
int Form_Wclose()
```
- Leírás:** A rutin a **Form\_Wsave()** függvénnyel mentett képernyőt állítja vissza.
- Visszatérés:** -1, ha a képernyőterület nem volt mentve, egyébként 0.
- Lásd még:** **Form\_Wclose()**

## Form\_First\_Text

- Funkció:** A form által meghatározott képernyő első szövegkonstansának átírása.
- Használat:**
- ```
#include <FORM.H>
int Form_First_Text(page,txt)
    int page;          a kívánt oldal
    char *txt;          az új szövegkonstans
```
- Leírás:** A rutin a megadott lap első szövegkonstansát állítja be a kívánt új értékre.
- Visszatérés:** -1, ha az új szövegkonstans nem fér el a lapon, vagy ha az adott oldalon nincs szövegkonstans, egyébként 0.
- Lásd még:** **Form_Last_Text()**, **Text_Crea()**

Form_Last_Text

Funkció: A form által meghatározott képernyő utolsó szövegkonstansának átírása.

Használat:

```
#include <FORM.H>
int Form_Last_Text(page,txt)
    int    page;    a kívánt oldal
    char    *txt;    az új szövegkonstans
```

Leírás: A rutin a megadott lap utolsó szövegkonstansát állítja be a kívánt új értékre.

Visszatérés: -1, ha az új szövegkonstans nem fér el a lapon, vagy ha az adott oldalon nincs szövegkonstans, egyébként 0.

Lásd még: Form_First_Text(), Text_Crea()

5.1.6. Ciklikus adatfeldolgozás

Az adatbevitel menete általában az M-FORM futtatórendszerének ciklikus hívásán alapszik. Az ebbe a csoportba tartozó függvények teszik lehetővé az adatkezelés vezérlését.

Form_Init

Funkció: A form mezőinek inicializálása.

Használat:

```
#include <FORM.H>
Form_Init()
```

Leírás: A buffer feltöltése és a Form_Exec() hívása előtt hívandó rutin, amely az egyes mezőket a következők szerint tölti fel:

- megmaradó értékű (CARRY) mező esetén meg hagyja a korábban kapott értéket, egyébként
- ha van default érték, akkor ezt tölti be, ha nincs, kitöltetlenre állítja a mező értékét.

Visszatérés: Nincs.

Lásd még: Field_Def_All(), Field_Def(), Field_Carry()

Form_Exec

Funkció: A form alapján az adatbevitel vezérlése.

Használat:

```
#include <FORM.H>
```

```
int Form_Exec(pressed)
```

```
int pressed; az előre leütött billentyű kódja
```

Leírás: Vezérli az aktuális form kitöltését.

Visszatérés: Egy billentyűkód, amely lehet:

ESC;

PGUP, PGDN (scan kód, tehát a FORM.H-ban definiált

SCAN() makróval értelmezhető);

a **Form_Break()** által beállított érték.

Lásd még: **Form_Break()**, **FORM.H**

Form_Break

Funkció: A **Form_Exec** működésének megszakítása.

Használat:

```
#include <FORM.H>
```

```
Form_Break(code)
```

```
int code; a Form_Exec() kívánt visszatérési értéke
```

Leírás: A **Form_Exec()** által meghívott felhasználói C függvényekből ezzel a rutinnal lehet a **Form_Exec()** végrehajtását megszakítani. A **Form_Exec()** visszatérési értéke a paraméter értéke lesz.

Visszatérés: Nincs.

Form_Changed

Funkció: Értékváltozás kezelése.

Használat:

```
#include <FORM.H>
```

```
Form_Changed(fieldnum, actrec)
```

```
int fieldnum; az aktuális mező sorszáma
```

```
int actrec; az aktuális alrekord sorszáma
```


Leírás:

Ez a többitől eltérő használatú rutin a FORM könyvtárban egy üres (semmit nem végző) rutin. Használatának az a módja, hogy a programozó kiváltja egy saját maga által írt rutinnal. A **Form_Changed()** rutinra akkor kerül a vezérlés, ha a **Form_Exec()** végrehajtása során valamelyik mező értéke ténylegesen megváltozott vagy egy alrekord törlődött. (Tehát ha az "XYZ" értékű mezőt változtatjuk, de a változtatás után a mező értéke újra "XYZ", akkor a rutin nem hívódik meg). Ezzel a programozó lehetőséget kap arra, hogy teljesen kezében tartsa a rekord megváltozását (pl.: az aktuális rekord csak akkor mentendő le, ha valamelyik mező értéke változott).

A paraméterek jelentése a következő:

fieldnum: az aktuális mező sorszáma, vagy -1, ha alrekord törlése miatt van változás;

actrec: ha SUBFORM mezőn belül vagyunk, az aktuális alrekord sorszáma, egyébként -1.

Visszatérés: Nincs.

Példa:

```
#include <FORM.H>

int    changed;

process()
{
    changed = 0;
    Form_Exec(0);
    if(changed)
        saverecord();
}

Form_Changed(fieldnum, actrec)

int    fieldnum;
int    actrec;
{
    changed = 1;
}
```

Field_Select

Funkció: A következő aktuális mező beállítása.

Használat:

```
#include <FORM.H>
int Field_Select(fieldnum)
int fieldnum; a kívánt mező sorszáma
```

Leírás: A **Form_Exec()** hívása előtt és a felhasználói C függvényekből ezzel a rutinnal lehet meghatározni a következő aktuális mezőt. Ha a *fieldnum* értéke -1, akkor az aktuális mező az első input mező lesz.

Visszatérés: -1, ha nincs ilyen sorszámú mező vagy a mező védett, egyébként az átadott paraméter.

Field_Def_All

Funkció: A form-hoz tartozó összes mező feltöltöttségének jelzése.

Használat:

```
#include <FORM.H>
Field_Def_All(onoff)
int onoff; ki- vagy bekapcsolás
```

Leírás: Jelzi a futtatórendszer számára, hogy a form összes mezője valamilyen (nem a futtatórendszer hatáskörébe tartozó) módszerrel kapott értéket, vagy értékét veszítette. A paraméter jelentése:

0; a mezőknek nincs értéke,
1; a mezők értéket kaptak.

Visszatérés: Nincs.

Lásd még: **Field_Def()**, **Field_Valued()**, **Field_No_Valued()**

Field_No_Valued

Funkció: Az első definiálatlan értékű mező sorszámának visszaadása.

Használat:

```
#include <FORM.H>
int Field_No_Valued()
```


Leírás: Visszaadja, hogy az aktuális form input mezői közül melyik az első, amelynek az értéke nem definiált. Elsősorban standard program írását könnyíti meg.

Visszatérés: -1, ha minden mező értéke definiált, egyébként az első definiálatlan értékű mező sorszáma.

Lásd még: `Field_Valued()`, `Field_Def()`

Példa:

```
process()
{
    int    f;
    again:
    Form_Exec(0);
    if((f = Field_No_Valued()) >= 0)
    {
        Field_Select(f);
        goto again;
    }
}
```

5.1.7. Mezőértékek karbantartása

A form-hoz tartozó mezők értékét a következő estekben kell karbantartani:

- az operátori kezelés előtt az aktuális érték beállítása,
- az operátori kezelés során az érték meghatározása,
- az operátori kezelés után az új érték kimentése.

Ezen kívül számos esetben meg kell változtatnunk programunkból a mezők értékét (pl. számított output mezők, összegző rovat stb.) A mezők értékének karbantartása a következő módokon mehet végbe:

- a futtatórendszer következő függvényeinek hívásával,
- a beolvasó vagy az ellenőrző függvényben a kapott címparaméter felhasználásával,
- a futtatórendszer belső bufferének felhasználásával direkt módon, vagy a `FormEdit` által definiált struktúra felhasználásával. A módszer esetén gondoskodni kell arról, hogy a mezők értékének beállítása után a

Field_Def() vagy a **Filed_Def_All()** függvénnyel jelezzük a futtatórendszer számára, hogy a mező értéket kapott.

A mezők értékének karbantartása a következő öt függvénnyel lehetséges.

Field_Set

Funkció: A mező értékének beállítása.

Használat:

```
#include <FORM.H>
```

```
int Field_Set(fieldnum,addr)
```

int *fieldnum*; a kívánt mező sorszáma

char **addr*; az input változó címe

Leírás: A megadott mezőt feltölti a második paraméter által meghatározott változóban található értékkel.

Visszatérés: -1, ha a mező formátumos, és az átadott érték nem felel meg a formátumnak (azaz a kötött karakterek helyén nem a megfelelő karakter áll).

Lásd még: **Field_F_Set()**

Field_F_Set

Funkció: A formátumos mező értékének beállítása.

Használat:

```
#include <FORM.H>
```

```
Field_F_Set( fieldnum,addr)
```

int *fieldnum*; a kívánt mező sorszáma

char **addr*; az input változó címe

Leírás: A megadott mezőt feltölti a második paraméter által meghatározott változóban található értékkel. Ha a mezőhöz formátum van rendelve, akkor az átadott érték nem tartalmazhatja a kötött karaktereket. A kötött karakterek a mező megfelelő helyeire automatikusan bekerülnek.

Visszatérés: Nincs.

Lásd még: **Field_Set()**

Field_Get

Funkció: A mező értékének kiolvasása.

Használat:

```
#include <FORM.H>
int Field_Get(fieldnum, addr )
    int fieldnum;    a kívánt mező sorszáma
    char *addr;      az output változó címe
```

Leírás: A megadott mező értékét a második paraméter által megadott címre tölti, ha a mező értéke definiált.

Visszatérés: -1, ha a mező értéke nem definiált, egyébként a mező sorszáma.

Lásd még: Field_F_Get()

Field_F_Get

Funkció: A formátumos mező tömörített értékének kiolvasása.

Használat:

```
#include <FORM.H>
Field_F_Get(fieldnum, addr)
    int fieldnum;    a kívánt mező sorszáma
    char *addr;      az output változó címe
```

Leírás: A megadott mező értékét a második paraméter által megadott címre tölti, ha a mező értéke definiált. Ha a mezőhöz formátum van rendelve, akkor a kötött karakterek nem kerülnek be az outputba.

Visszatérés: Nincs.

Lásd még: Field_Get()

Field_Edit

Funkció: A mező beolvasása.

Használat:

```
#include <FORM.H>
int Field_Edit(fieldnum, pressed)
    int fieldnum;    a kívánt mező sorszáma
```

int *pressed*; az előre leütött billentyű kódja

Leírás: A megadott mező értékének a beolvasását és (ellenőrző függvény megléte esetén) az ellenőriztetését végzi.

Visszatérés: A kiszálló billentyű kódja (ESC, ENTER, UP, DOWN, RIGHT, PGUP, PGDN).

Lásd még: Edit_Num(), Edit_Frm()

5.1.8. Mezőjellemzők lekérdezése

A program futása során szükség van az egyes mezők jellemzőinek dinamikus lekérdezésére (hiszen így valósítható meg, hogy hasonló mezőket azonos rutinokkal kezeljünk). A következő függvények ezt a célt szolgálják.

Field_Act

Funkció: Az aktuális mező sorszámának visszaadása.

Használat:

```
#include <FORM.H>
```

```
int Field_Act()
```

Leírás: Visszaadja az aktuális mező sorszámát.

Visszatérés: Az aktuális mező sorszáma.

Field_Col

Funkció: A mező kezdőoszlopának visszaadása.

Használat:

```
#include <FORM.H>
```

```
int Field_Col(fieldnum)
```

```
int fieldnum;      a kívánt mező sorszáma
```

Leírás: Visszaadja a mező kezdőpozícióját.

Visszatérés: A mező oszlopa.

Lásd még: Field_Row(), Field_Length()

Field_Dec

Funkció: Típustól függő információ visszaadása.

Használat:

```
#include <FORM.H>
```

```
int Field_Dec(fieldnum)
```

```
int fieldnum; a kívánt mező sorszáma
```

Leírás:

A visszatérési érték a mező típusától függően:

BYTE,INT,LONG esetén 0, ha az egész mező előjeles;

FLOAT,DOUBLE esetén a tizedesjegyek száma;

DATE esetén a dátum formája (0: angol, 1: magyar,
2: német);

USER és SUBFORM esetén a mező jobb oszlopa.

Visszatérés: Lásd a leírást!

Field_Is_Carry

Funkció: A mező megmaradó értékűségének vizsgálata.

Használat:

```
#include <FORM.H>
```

```
int Field_Is_Carry(fieldnum)
```

```
int fieldnum; a kívánt mező sorszáma
```

Leírás:

Visszaadja, hogy a mező megmaradó értékű-e. Ha igen, akkor a **Form_Init()** függvény meghagyja az előző értéket (ha volt).

Visszatérés: 0, ha a mező nem tartja meg az értékét; 1, ha a mező a **Form_Init()** hívásakor megtartja az értékét.

Lásd még: **Field_Carry()**

Field_Is_Prot

Funkció: A mező védettségének vizsgálata.

Használat:

```
#include <FORM.H>
```

```
int Field_Is_Prot(fieldnum)
```

```
int fieldnum; a kívánt mező sorszáma
```

Leírás: Visszaadja, hogy a mező védett-e.
Visszatérés: 0, ha a mező nem védett (input); 1, ha a mező védett (output).
Lásd még: `Field_Protect()`

Field_Length

Funkció: A mező képernyőbeli hosszának visszaadása.

Használat:

```
#include <FORM.H>
int Field_Length(fieldnum)
    int fieldnum;    a kívánt mező sorszáma
```

Leírás: Visszaadja a mezőnek a képernyőn elfoglalt pozícióinak számát. USER és SUBFORM típusú mezők esetén ez a mező sorainak száma.

Visszatérés: A mező képernyőbeli hossza.

Lásd még: `Field_Size()`

Field_List

Funkció: Teljes képernyős mezőkiválasztás.

Használat:

```
#include <FORM.H>
int Field_List(title,def,fvp)
    char    *title;    a kiválasztó menü címe
    int     def;        az alapértelmezett mező sorszáma
    int     (*fvp)();    ENTER esetén végrehajtandó fv.
```

Leírás: A rutin kiírja egy dinamikus ablakba az aktuális form mezőinek nevét bekeretezve, és a megadott címmel ellátva. A kurzorvezérlő billentyűkkel való kiválasztás után az ENTER leütésére a következő történik:

- Ha nincs meghívandó függvény (az *fvp* függvénytató értéke 0), akkor visszaadja a kiválasztott mező sorszámát.
- Ha van meghívandó függvény, akkor ez meghívódik és paraméterként kapja a kiválasztott mező sorszámát. Ha a függvény visszatérési értéke nem 0, akkor a `Field_List()` visszaadja az aktuális mező sorszámát, egyébként a kiválasztás folytatódhat.

Visszatérés: -1 ESC leütése esetén, egyébként a kiválasztott mező sorszáma.

Field_Name

Funkció: A mező nevének visszaadása.

Használat:

```
#include <FORM.H>
char *Field_Name(fieldnum)
    int fieldnum;    a kívánt mező sorszáma
```

Leírás: Visszaadja az aktuális form megadott mezőjének a nevét.

Visszatérés: A mező neve.

Lásd még: Field_Sernum()

Field_Offset

Funkció: A mező belső bufferén belüli pozíciójának visszaadása.

Használat:

```
#include <FORM.H>
int Field_Offset(fieldnum)
    int fieldnum;    a kívánt mező sorszáma
```

Leírás: Visszaadja a megadott mezőnek a futtatórendszer belső bufferének kezdetétől számított kezdőpozícióját.

Visszatérés: A belső bufferbeli relatív pozíció.

Lásd még: Field_Size()

Field_Row

Funkció: A mező sorának visszaszámítása.

Használat:

```
#include <FORM.H>
int Field_Row(fieldnum)
    int fieldnum;    a kívánt mező sorszáma
```

Leírás: Visszaadja a mező képernyőbeli sorát.

Visszatérés: A mező sora.

Lásd még: Field_Row(), Field_Length()

Field_Sernum

Funkció: A megadott nevű mező megkeresése a mezők között.

Használat:

```
#include <FORM.H>
int Field_Sernum(name)
char *name;    a kívánt mező neve
```

Leírás: Visszaadja a megadott nevű mező sorszámát.

Visszatérés: -1, ha nincs ilyen nevű mező, egyébként a mező sorszáma.

Lásd még: Field_Name()

Field_Size

Funkció: A mező belső méretének visszaadása.

Használat:

```
#include <FORM.H>
int Field_Size(fieldnum)
int fieldnum;    a kívánt mező sorszáma
```

Leírás: Visszaadja a megadott mező bufferben lefoglalt helyének méretét byte-ban.

Visszatérés: A mező belső mérete byte-ban.

Lásd még: Field_Offset(), Field_Length()

Field_Type

Funkció: A mező típusának visszaadása.

Használat:

```
#include <FORM.H>
int Field_Type(fieldnum)
int fieldnum;    a kívánt mező sorszáma
```

Leírás: Visszaadja a megadott mező típusát. A típus a <FORM.H> header-file-ban definiáltak szerint értelmezhető:

FLD_BYTE	1-byte-os egész;
FLD_INT	2-byte-os egész;
FLD_LONG	4-byte-os egész;
FLD_FLOAT	4-byte-os valós;

FLD_DOUBLE	8-byte-os valós;
FLD_DATE	dátum négy byte-on;
FLD_TEXT	a mező hosszának megfelelő méretű karaktertömb;
FLD_USER	USER mező, tetszőleges típus (1 egész érték);
FLD_SUBF	SUBFORM mező (1 egész érték).

Visszatérés: A mező típusa.

Field_Valued

Funkció: A mező feltöltöttségének vizsgálata.

Használat:

```
#include <FORM.H>
```

```
int Field_Valued(fieldnum)
```

```
int fieldnum;    a kívánt mező sorszáma
```

Leírás: Visszaadja, hogy a mező a **Form_Init()** hívása óta kapott-e értéket. Egy mező a következő 6 módon kaphat értéket:

1. inicializálással – **Field_Set()**;
2. kezdeti érték (default) beállításával – **Form_Init()**;
3. megmaradó értékű mezőnél a korábban kapott érték meg-hagyásával – **Field_Carry()**, **Form_Init()**;
4. normál adatbevitellel – **Form_Exec()**, **Field_Edit()**;
5. memória másolással – **Form_Buffer()**,
Form_Size(), **memcpy()**;
6. a header-file-ban definiált struktúra felhasználásával (érté-kadás).

Mivel az 5. és 6. módszer nem tartozik a futtatórendszer ha-táskörébe, a mező feltöltöttségét jelezni kell a futtatórend-szernek.

A feltöltöttség a **Field_Def()** és a **Field_Def_All()** függvé-nyekkel jelezhető.

Visszatérés: 0, ha a mező nem kapott értéket, 1, ha a mező az első négy mód valamelyikével értéket kapott, vagy ha a **Field_Def()**, ill. **Field_Def_All()** függvények ezt jelezték a futtatórend-szernek.

Lásd még: **Field_No_Valued()**, **Field_Def()**, **Field_Def_All()**

5.1.9. Mezőjellemzők beállítása

A következő rutinok az egyes mezők olyan jellemzőit állítják át, ill. be, amelyek az adatbevitel menetétől függően változhatnak.

Field_Carry

Funkció: A mező megmaradó értékűségének állítása.

Használat:

```
#include <FORM.H>
```

```
Field_Carry(fieldnum, onoff)
```

int *fieldnum*; a kívánt mező sorszáma

int *onoff*; ki- vagy bekapcsolás

Leírás: Lehetőség van a mezők alapértelmezett értékének vagy "nem definiált" értékének automatikus felülírására a mező előző értékével:

- Ha a rutin második paramétere 0, akkor a "carry" funkció kikapcsolódik, tehát a következő **Form_Init()** végrehajtásakor a mezőben a default érték jelenik meg (vagy ennek hiányában nem definiált).
- Ha a második paraméter nem 0, akkor a "carry" funkció bekapcsolódik, tehát a következő **Form_Init()** végrehajtásakor a mezőben a korábban kapott érték marad meg.

Visszatérés: Nincs.

Lásd még: **Field_Is_Carry()**

Field_Def

Funkció: A mező feltöltöttségének jelzése.

Használat:

```
#include <FORM.H>
```

```
Field_Def(fieldnum, onoff)
```

int *fieldnum*; a kívánt mező sorszáma

int *onoff*; ki- vagy bekapcsolás

Leírás: Jelzi a futtatórendszer számára, hogy a megadott mező értéke valamilyen (nem a futtatórendszer hatáskörébe tartozó) módszerrel definiálatlanná vagy definiálttá vált.

Visszatérés: Nincs.

Lásd még: **Field_Def_All()**, **Field_Valued()**, **Field_No_Valued()**

Field_Protect

Funkció: A mező védettségének beállítása.

Használat:

```
#include <FORM.H>
```

```
Field_Protect(fieldnum,onoff)
```

```
int fieldnum;    a kívánt mező sorszáma
```

```
int onoff;        ki- vagy bekapcsolás
```

Leírás: Beállítja a megadott mező védettségét a második paraméter alapján. Ha a paraméter 0, akkor a mező a rutin hívása után input mező (nem védett), egyébként output mező.

Visszatérés: Nincs.

Lásd még: **Field_Is_Protect()**

SubForm_Set

Funkció: Egy SUBFORM típusú mező kezelési jellemzőinek beállítása.

Használat:

```
#include <FORM.H>
```

```
int SubForm_Set(fieldnum,frst,act)
```

```
int fieldnum;    a kívánt SUBFORM mező sorszáma
```

```
int frst;         melyik legyen az első kiírt rekord
```

```
int act;         melyik legyen az aktuális rekord
```

Leírás: A függvény segítségével beállíthatjuk, hogy a SUBFORM mezőben melyik legyen az első kiírt alrekord (ha az összes alrekord nem fér el a SUBFORM mezőhöz rendelt képernyőn), és, hogy melyik legyen az aktuális alrekord.

Visszatérés: -1, ha a mező nem SUBFORM, egyébként 0.

5.1.10. Standard beolvasó függvények

A mezők értékének billentyűzetről való beolvasása a futtatórendszer két rutinjára vezethető vissza:

- **EDIT_Num()**: numerikus adatok beolvasása;
- **Edit_Fmt()**: karakteres, formátummal vezérelt beolvasás.

E két függvény ismertetése elsősorban a felhasználó saját beolvasó függvényének megírását könnyíti.

Edit_Fmt

Funkció: Formátumos mező szerkesztő.

Használat:

```
#include <FORM.H>
#include <KGET.H>
#include <SETATTR.H>
int Edit_Fmt( buf,len,clr,pressed,format)
char    *buf;           a szerkesztendő szöveg buffere
int     len;            a szerkeszthető hossz
int     clr;            a szerkesztési szín
int     pressed;        az előre leütött billentyű kódja
char    *format;        a formátum
```

Leírás: A rutin egy mezőeditor, amely a képernyő aktuális pozíciójától kezdve a megadott hosszon a buf bufferben lévő szövegnek a formátum szerinti szerkesztését teszi lehetővé.

Visszatérés: -1 hibás paraméter esetén, egyébként a kiszálló billentyű kódja (ESC, ENTER, UP, DOWN, RIGHT, PGUP, PGDN).

Lásd még: Edit_Num()

Edit_Num

Funkció: Numerikus string szerkesztése.

Használat:

```
#include <FORM.H>
#include <KGET.H>
#include <SETATTR.H>

int Edit_Num( buf,len,clr,pressed,isuns,isfloat)
char    *buf;           a szerkesztendő szöveg buffere
int     len;            a szerkeszthető hossz
int     clr;            a szerkesztési szín
int     pressed;        az előre leütött billentyű kódja
int     isuns;          előjel nélküli érték?
int     isfloat;        tizedespont engedélyezett?
```


- Leírás:** A rutin egy mezőeditor, amely a képernyő aktuális pozíciójától kezdve a megadott hosszon a *buf* bufferben lévő numerikus string szerkesztését teszi lehetővé.
- Visszatérés:** -1 hibás paraméter esetén, egyébként a kiszálló billentyű kódja (ESC, ENTER, UP, DOWN, RIGHT, PGUP, PGDN).
- Lásd még:** `Edit_Fmt()`

5.1.11. Formátum kezelés

A következő függvények formátummal vezérelt karakteres mezők kezelését könnyítik meg.

Check_Fmt

Funkció: Megvizsgálja, hogy a szöveg megfelel-e a formátumnak.

Használat:

```
#include <FORM.H>
int Check_Fmt(fmt,src,len)
char    *fmt;    a formátum stringje
char    *src;    a forrás szöveg
int     len;     a kívánt hossz
```

Leírás: Megvizsgálja, hogy a szöveg megfelel-e a formátumnak.

Visszatérés: -1, ha a szöveg nem felel meg a formátumnak, 0 egyébként.

Compress_Fmt

Funkció: A formátumos szöveget a kötött karakterek nélkül másolja.

Használat:

```
#include <FORM.H>
Compress_Fmt(fmt,src,dest,len)
char    *fmt;    a formátum stringje
char    *src;    a forrás szöveg
char    *dest;   az output buffer
int     len;     a kívánt hossz
```

Leírás: A formátumos szöveget a kötött karakterek nélkül másolja.

Visszatérés: Nincs.

Lásd még: `Extract_Fmt()`

Copy_Fmt

Funkció: Az input szöveg formátumnak megfelelő átmásolása.

Használat:

```
#include <FORM.H>
```

```
Copy_Fmt(fmt,src,dest,len)
```

char *fmt; a formátum stringje

char *src; a forrás szöveg

char *dest; az output buffer

int len; a kívánt hossz

Leírás: A rutin a forrás szöveget másolja át az output bufferbe úgy, hogy a kötött karakterek pozícióján lévő karaktereket nem figyeli, és a formátumnak nem megfelelő karakterek helyett szóközőket tesz.

Visszatérés: Nincs.

Lásd még: Extract_Fmt(), Fill_Fmt()

Extract_Fmt

Funkció: A kötött karaktereket nem tartalmazó formátumos szövegbe a kötött karakterek beillesztése.

Használat:

```
#include <FORM.H>
```

```
Extract_Fmt(fmt,src,dest,len)
```

char *fmt; a formátum stringje

char *src; a forrás szöveg

char *dest; az output buffer

int len; a kívánt hossz

Leírás: A kötött karaktereket nem tartalmazó formátumos szövegbe beilleszti a kötött karaktereket.

Visszatérés: Nincs.

Lásd még: Compress_Fmt()

Fill_Fmt

Funkció: Formátumos mező inicializálása.

Használat:

```
#include <FORM.H>
```

```
Fill_Fmt(fmt,dest,len,fill)
```

```
char    *fmt;    a formátum stringje
```

```
char    *dest;   az output buffer
```

```
int      len;    a kívánt hossz
```

```
int      fill;   a kitöltő karakter
```

Leírás: Az output buffert kitölti a kötött karakterekkel és a megadott (*fill*) karakterrel (rendszerint szóköz).

Visszatérés: Nincs.

5.1.12. Dátumkezelés

Az M-FORM az angol, a magyar és a német dátum formát támogatja:

angol: HH/NN/EE

magyar: EE/HH/NN

német: NN/HH/EE

Belső ábrázolásuk közös. A dátumot a következő rutinok kezelik:

atodat(): dátumot tartalmazó szöveg konvertálása belső formába.

convdato(): belső ábrázolású dátum konvertálása a megfelelő karakteres formába.

sysdate(): az aktuális dátum lekérdezése.

atodat

Funkció: Karakteres dátum konverziója értékké.

Használat:

```
#include <FORM.H>
```

```
long atodat(str,format)
```

```
char    *str;    a karakteres dátum
```

```
int      format;    dátum forma
```

Leírás: Ellenőrzéssel konvertálja a második paraméter által meghatározott formában megadott dátumot egy 4-byte-os értékké. Az érték kezelése a FORM.H header-file-ban definiált DATE struktúrával és az UDATE union-nal kezelhető.

A dátum formája a következők valamelyike lehet:

0 hh/nn/ee (angol forma);

1 ee/hh/nn (magyar forma);

2 nn/hh/ee (német forma).

Visszatérés: -1, rossz dátum esetén, egyébként a dátum értéke.

Lásd még: convdat()

convdat

Funkció: A belső ábrázolású (tömörített) dátum konverziója karakteres formába.

Használat:

```
#include <FORM.H>
```

```
char *convdat(date,format)
```

```
long   date;           a tömörített dátum
```

```
int     format;         dátum forma
```

Leírás: A belső ábrázolású dátumot (amely a FORM.H header-file-ban definiált DATE struktúrával és az UDATE union-nal kezelhető) karakteres formába konvertálja. A konverzió egy statikus bufferben történik, amely a következő convdat hívásig megőrzi értékét.

Visszatérés: A megfelelő formájú dátumstring pointere.

Lásd még: atodat()

sysdate

Funkció: A rendszerdátum betöltése változóba.

Használat:

```
#include <FORM.H>
```

```
long sysdate(addr)
```

```
struct DATE *addr; dátum típusú mező címe
```

Leírás: A megadott 4-byte-os címre betölti a rendszer dátumát. Ha az *addr* NULL értékű, akkor a rutin csak a visszatérési értékkel adja meg a dátumot. A DATE struktúra és az UDATE union a <FORM.H> header-file-ban van definiálva.

Visszatérés: A rendszer dátuma LONG értéként.

5.1.13. Konverziók

Az M-FORM könyvtára hat további konverziós rutint tartalmaz, kettőt az előjel nélküli numerikus értékek, kettőt a C stringek, kettőt pedig a leütött billentyű nevének kezelésére.

atolu

Funkció: String átalakítása belső ábrázolású előjel nélküli unsigned long értékre.

Használat:

```
#include <FORM.H>
unsigned long atolu(str)
char *str; a C string
```

Leírás: Az átadott stringet előjel nélküli unsigned long értékre konvertálja.

Visszatérés: A string értéke.

Lásd még: atol(), atoi(), atou()

atou

Funkció: String átalakítása belső ábrázolású előjel nélküli unsigned int értékre.

Használat:

```
#include <FORM.H>
unsigned int atou(str)
char *str; a C string
```

Leírás: Az átadott stringet előjel nélküli unsigned int értékre konvertálja.

Visszatérés: A string értéke.

Lásd még: atol(), atoi(), atolu()

strtotxt

Funkció: String átalakítása belső ábrázolású szöveges mező formába.

Használat:

```
#include <FORM.H>
strtotxt(str,txt,len)
char    *str;    C string
char    *txt;    belső szövegmező címe
int     len;     belső szövegmező hossza
```

Leírás: A C nyelvben jól kezelhető stringeket az M-FORM futtatórendszere nem tudja kezelni. A stringeknek a belső ábrázolású karaktertömbbe való konvertálását végzi az **strtotxt()** függvény úgy, hogy a stringet szóközzel a szükséges méretre kipótálja, és a lezáró 0 nélkül, a *txt* paraméterrel meghatározott területre másolja.

Visszatérés: Nincs.

Lásd még: **txttostr()**

txttostr

Funkció: Szöveg típusú mező konverziója C stringbe.

Használat:

```
#include <FORM.H>
txttostr(txt,len,str)
char    *txt;    a belső ábrázolású szöveg
int     len;     a szöveg hossza
char    *str;    az output string helye
```

Leírás: A futtatórendszer a szöveg típusú mezőket fix hosszon, és 0 záró karakter nélkül kezeli. Ez a függvény a mező tartalmát az *str* paraméterként átadott területre, a C string konvencióknak megfelelő formában áttölti. A mező végén lévő szóközt a rutin levágja, és a karaktersort 0-val zárja le. Az *str* paraméterrel átadott terület hossza legalább *len*+1 (záró 0)!

Visszatérés: Nincs.

Lásd még: **strtotxt()**

Key_Name

Funkció: Egy billentyűkód szabványosított nevének visszaadása.

Használat:

```
#include <FORM.H>
char *Key_Name(code)
    unsigned code; a billentyű ASCII + scan kódja
```

Leírás: Az átadott billentyűkódnak megfelelő nevet tartalmazó statikus karaktertömb címét adja vissza.

Visszatérés: A billentyű neve.

Lásd még: Key_Scan()

Key_Scan

Funkció: Egy szabványosított billentyűnév alapján az ASCII + scan kód képzése.

Használat:

```
#include <FORM.H>
unsigned Key_Scan(kname)
    char *kname; a billentyű neve
```

Leírás: Az átadott billentyűnévnek megfelelő ASCII + scan kódot adja vissza.

Visszatérés: A billentyű kódja.

Lásd még: Key_Name()

5.2. Felhasználói függvények

A felhasználói C függvények a programozó által megírandó függvények, amelyek csatlakozási pontjait a FormEdit használatakor kell meghatározni.

A felhasználói függvények a következők lehetnek:

- billentyűhöz rendelt függvény (opcionális),
- beolvasó (input) függvény (opcionális, kivéve a USER típusú mezők-nél),

- ellenőrző, aktualizáló (update) függvény (opcionális),
- kiíró (screen) függvény USER típusú mezőknél,
- betöltő (load) függvény SUBFORM típusú mezőknél,
- kimentő (store) függvény SUBFORM típusú mezőknél.

A felhasználói függvényekkel kapcsolatban általános szabályként elmondható, hogy

- a beépített felhasználói függvények az adatbevitel során bizonyos jól meghatározott esetekben hívódnak,
- szigorúan be kell tartani a (következőkben meghatározott) paraméterezési és visszatérési konvenciókat,
- a beépített felhasználói függvények az aktív form-mal és az amögött álló adathalmazzal a futtatórendszer rutinjain keresztül tudják tartani a kapcsolatot.

A felhasználói függvények megírásához nagy segítséget nyújt a FormEdit Alt-G (generate C functions) funkciója, amely egy .FCC típusú file-ban generálja az összes felhasználói C függvény forrását a paraméterlistával, magyarázattal ellátva. A függvények megírását e forrás másolásával kezdhetjük.

A generálást a FormEdit program könyvtárában lévő FORMEDxx.SRF (Source of Referenced Function) prototípus file-ok alapján képzi. Ezek a file-ok módosíthatók, amennyiben a programozó általánosan alkalmazható, más prototípust akar. (Természetesen több felhasználó esetén a módosításokat egyeztetni kell!)

Az egyes SRF file-ok a következő prototípusokat tartalmazzák:

FORMED00.SRF	input függvény, BYTE típus
FORMED01.SRF	update függvény, BYTE típus
FORMED10.SRF	input függvény, INT típus
FORMED11.SRF	update függvény, INT típus
FORMED20.SRF	input függvény, LONG típus
FORMED21.SRF	update függvény, LONG típus
FORMED30.SRF	input függvény, FLOAT típus
FORMED31.SRF	update függvény, FLOAT típus
FORMED40.SRF	input függvény, DOUBLE típus
FORMED41.SRF	update függvény, DOUBLE típus
FORMED50.SRF	input függvény, DATE típus
FORMED51.SRF	update függvény, DATE típus
FORMED60.SRF	input függvény, TEXT típus
FORMED61.SRF	update függvény, TEXT típus
FORMED70.SRF	input függvény, USER típus

FORMED71.SRF	screen függvény, USER típus
FORMED80.SRF	load függvény, SUBFORM típus
FORMED81.SRF	store függvény, SUBFORM típus
FORMED99.SRF	billentyűhöz rendelt függvény

A következőkben leírjuk a futtatórendszer által hívott felhasználói függvények illesztésének szabályait.

Az itt közölt függvénynevek természetesen nem a használandó nevek, csak a függvény természetére utalnak.

5.2.1. Beolvasó függvény

A mező beolvasó függvényeket a FormEdit-en belül az egyes mezők definiálásánál vagy módosításánál lehet meghatározni. Ha ez nem definiált, akkor a futtatórendszer a megfelelő konvertálás után a standard beolvasó függvényt hívja meg (**Edit_Num()** vagy **Edit_Fmt()**). Mivel a USER mező beolvasó függvénye mind funkciójában, mind használatában eltér a többi beolvasó függvénytől, ezért leírása az 5.2.3. "USER mező beolvasó függvénye" című pontban található.

Használat: nem USER típusú mezőknél:

```
#include <kbget.h>
#include <FORM.H>
int fld_input(fieldnum,addr,length,dec,pressed)
    int    fieldnum;      a mező sorszáma
    TYP    *addr;         a mező form buffer-beli kezdőcíme
    int    length;        a mező hossza a képernyőn
    int    par;           típustól függő paraméter
    int    clr;           az editálási szín
    int    pressed;       a már leütött billentyű kódja
```

Az input függvény a mező típusától függően más paraméterezést kap:

- A második paraméter, amely a form bufferén belüli címet jelenti, a megfelelő típusú pointer.
- A negyedik paraméter jelentése:
 - BYTE, INT és LONG típusoknál: logikai érték, amely 0, ha a mező előjeles és 1, ha a mező előjel nélküli egész.
 - FLOAT és DOUBLE típusoknál a decimális jegyek száma.

- DATE típusnál a dátum formátuma: 0 angol, 1 magyar, 2 német.
- TEXT típusnál a paraméternek nincs jelentése,

A *pressed* értéket mint első leütött billentyűt kell kezelnie a felhasználói függvénynek. Ezt a **Form_Exec()** természetesen akkor sem írja ki a képernyőre, ha ez egy, a mező típusnak megfelelő karakter!

A függvénynek az ESC billentyűre a mező változtatás nélküli elhagyásával kell reagálnia.

A függvény visszatérési értéke a következő vezérlési billentyűk kódja lehet (ezek közül az ESC jelenti a mező változtatás nélküli elhagyását, a többi a változtatás befejezését, és az ezután következő tevékenységet határozza meg):

```
ESC
ENTER
CTR_A
SCAN(UP)
SCAN(DOWN)
SCAN(LEFT)
SCAN(RIGHT)
SCAN(PGUP)
SCAN(PGDN)
SCAN(HOME)
SCAN(END)
```

A fenti értékek a CEX könyvtárhoz tartozó KBGET.H header-file-ban vannak definiálva, a SCAN() makró a FORM.H-ban definiált.

Az alábbi példa egy három karakter hosszú, TEXT típusú mező beolvasó függvénye, amellyel a mezőnek a 12 hónap valamelyikének hárombetűs angol nyelvű rövidítését lehet rendelni:

```
#include <kbget.h>
#include <FORM.H>
char *MONTHS[] = {
    "Jan", "Feb", "Mar", "Apr",
    "May", "Jun", "Jul", "Aug",
    "Sep", "Oct", "Nov", "Dec",
    NULL};

int month_input(fieldnum, addr, length, dummy, pressed)
int fieldnum; /* a mező sorszáma */
char *addr; /* a mező form buffer-beli kezdőcíme */
```



```

int    length; /* a mező hossza a képernyőn */
int    dummy;  /* TEXT típusnál nincs jelentése */
int    clr;     /* az editálási szín          */
int    pressed; /* a már leütött billentyű kódja */
{
    int    val;
    val = selstr(MONTHS, "Months", 0);
    if(val < 0) /* kiválasztás visszautasítva */
        return(ESC);
    movbytes(addr, MONTHS[val], 3); /* a három betű bemásolása */
    return(ENTER); /* rendben, következő mező */
}

```

Megjegyzés: A CEX2 könyvtár `selstr()` függvényének három paramétere van:

1. a string tömb NULL-val lezárva (char **),
2. a menü keretébe kerülő cím (char *) és
3. a default érték (int).

A rutin a CEX2 szokásos 'gcolors' színpaletta CMSG (üzenet) színét használja.

5.2.2. Ellenőrző függvény

A mező elhagyásakor meghívandó felhasználói függvény többek között ellenőrzést, vagy az aktuális mezőtől függő további mezők értékének megváltoztatását végezheti.

Ezeket a függvényeket a FormEdit-en belül az egyes mezők definiálásánál vagy módosításánál lehet meghatározni.

A függvényt a `Form_Exec()` rutin hívja meg a mező beolvasása után akkor, ha a mező megadását nem az ESC billentyűvel fejeztük be.

Használat:

```

#include <FORM.H>
int fld_input(fieldnum, addr)
    int    fieldnum; /* a mező sorszáma */
    TYP    *addr;    /* a mező form buffer-beli
                     kezdőcíme */

```


A TYP (a beolvasó függvény 2. paraméteréhez hasonlóan) a mezőnek megfelelő C típust jelenti.

A függvény visszatérési értéke a következők valamelyike lehet (definíciójuk a <FORM.H> header-file-ban található):

F_UPD_OK	az érték jó, a mező tartalma megváltozott;
F_UPD_BAD	az érték rossz, változás nem történt;
F_UPD_AGAIN	az érték rossz, a mezőt újra ki kell tölteni.

FONTOS megjegyzés:

Ha az ellenőrző függvény az **F_UPD_OK** értékkel tér vissza, akkor a visszatérés előtt a függvénynek meg kell hívnia a **Field_Def()** függvényt. A **Field_Def()** függvénynek a saját *fieldnum* paramétert és az *1* értéket kell átadni (jelezvén a futtatórendszer számára, hogy a mező új értéket kapott).

A következő példa a magyar személyi szám (11 számjegy)

#####

formátumos beolvasása után ellenőrzi azt, majd helyes érték esetén beállítja a "SzulNap" mező értékét is a személyi számban található dátum alapján.

```
#include <FORM.H>
```

```
int SzemSzam_update(fieldnum, addr)
```

```
int fieldnum; /* a mező sorszáma */
```

```
char *addr; /* a mező form buffer-beli kezdőcíme */
```

```
{
```

```
register int i; /* ciklusváltozó */
```

```
register char *s; /* munkapointer */
```

```
int sum; /* a sz.sz. számjegyeinek  
összege */
```

```
char buf[9]; /* a sz.sz.-ból létrehozott  
dátum-string*/
```

```
long dat; /* a konvertált dátum */
```

```
long b; /* a 'SzulNap' mező értéke */
```

```
s = addr;
```

```
if(*s < '1' || *s > '4')
```

```
{
```

```
sherror("Rossz személyi szám!", NULL);
```

```
return(F_UPD_BAD);
```



```

    }
    sum = 0;
    for(i = 1; i < 11; i++)
        sum += (*s++-'0') * i; /* a számjegyek összegzése */
    if(*s-'0' != (sum % 11)) /* az ellenőrző számjegy */
    {
        sherror("Rossz személyi szám!",NULL);
        return(F_UPD_BAD);
    }
    /* a dátum string létrehozása a személyi számban lévő
       dátumból */
    strcpy(buf," / / ");
    s = addr;
    buf[0] = *++s;
    buf[1] = *++s;
    buf[3] = *++s;
    buf[4] = *++s;
    buf[6] = *++s;
    buf[7] = *++s;
    dat = atodat(buf,1); /* konvertálás */
    if(Field_Get(F_SzulNap,(char *)&b) < 0)
    {
        /* a mezőnek még nem volt értéke*/
        Field_Set(F_SzulNap,(char *) &dat);
        Field_Put(F_SzulNap);
        Field_Def(fieldnum,YES);/* a mező értéket kapott */
        return(F_UPD_OK); /* az érték elfogadva */
    }
    if(dat == b) /* egyeznek? */
    {
        Field_Def(fieldnum,YES);/* a mező értéket kapott */
        return(F_UPD_OK); /* az érték elfogadva */
    }
    sherror("A születésnap és a személyi szám!",NULL);
    return(F_UPD_BAD); /* az érték nincs elfogadva */
}

```

5.2.3. USER mező beolvasó függvénye

A USER mező beolvasó függvénye a következőkben különbözik a többi típus beolvasó függvényétől:

- Kötelező megadni, hiszen a futtatórendszer nem adhat megoldást erre a speciális esetre.
- A függvény akkor is meghívódik, ha olyan USER mezőre állunk, amelynek területe csak szököket tartalmaz (tehát valószínűleg nem definiált az értéke). Ez lehetőséget ad például arra, hogy ha a mező még nem definiált, akkor a mezőre való ráálláskor automatikusan megjelenjen a mező lehetséges értékeit felkínáló kiválasztási menü.
- Paraméterként kapja a képernyőn való elhelyezkedést.
- Az értékadást a **Field_Def()** függvénnyel a futtatórendszer számára jelezni kell.

Használat:

```
#include <kbget.h>
#include <FORM.H>

int user_input(fieldnum,ulr,ulc,lrr,lrc,clr,addr,pressed)

int    fieldnum;        /* a mező sorszáma*/
int    ulr;             /* a mező felső sora*/
int    ulc;             /* a mező bal oszlopa*/
int    lrr;             /* a mező alsó sora*/
int    lrc;             /* a mező jobb oszlopa*/
int    clr;             /* az editálási szín*/
int    *addr;           /* a mező form buffer-beli
                        kezdőcíme*/

int    pressed;         /* a már leütött billentyű kódja*/
```

A függvény lehetséges visszatérési értéke ugyanaz lehet, mint a többi beolvasó függvényénél (lásd az 5.2.1. "A beolvasó függvény" című pontot).

A példát lásd az előző pontban!

5.2.4. Kiíró függvény

A USER típusú mezők értékét a futtatórendszer a mezőhöz kötelezően hozzárendelt kiíró függvény hívásával jeleníti meg.

Használat:

```
#include <kbget.h>
#include <FORM.H>
int user_input(fieldnum,ulr,ulc,lrr,lrc,clr,addr)
int    fieldnum;      /* a mező sorszáma */
int    ulr;           /* a mező felső sora */
int    ulc;           /* a mező bal oszlopa */
int    lrr;           /* a mező alsó sora */
int    lrc;           /* a mező jobb oszlopa */
int    clr;           /* az editálási szín */
int    *addr;         /* a mező form buffer-beli
                      kezdőcíme */
```

A USER típusú mező használatának tipikus esete a kiválasztási menü alapján való értékadás. A következő példa azt mutatja, hogy négy fajta minőséget jelző mező hogyan kaphat értéket a CEX2 selstr() rutinjának segítségével, és ez hogyan kerülhet a képernyőre (tehát a USER mező beolvasó és kiíró függvényei következnek).

```
#include <kbget.h>
#include <FORM.H>
char *MINOSEGEK[] = {
    "Rossz",
    "Közepes",
    "Jó",
    "Kiváló",
    NULL};

int min_input(fieldnum, ulr, ulc, lrr, lrc, clr, addr,
              pressed)

int    fieldnum;      /* a mező sorszáma      */
int    ulr;           /* a mező felső sora      */
```

```

int    ulc;          /* a mező bal oszlopa */
int    lrr;          /* a mező alsó sora */
int    lrc;          /* a mező jobb oszlopa */
int    clr;          /* az editálási szín */
int    *addr;        /* a mező form buffer-beli
                      kezdőcíme */

int    pressed;      /* a már leütött billentyű kódja */

{
int    new;          /* az új érték */
new = selstr(MINOSEGEK, "Minőségek", *addr);
if(new < 0)          /* kiválasztás visszautasítva */
{
Field_Select(fieldnum+1); /* ugrás a
                           következő mezőre */
return(ESC); /* értékadás nem történt */
}
*addr = new;         /* az érték bekerül a form
                      bufferébe */
Field_Def(fieldnum, 1); /* jelzés az értékadásról */
return(SCAN(RIGHT)); /* ugrás a következő
                      mezőre */
}

int min_screen(fieldnum, ulr, ulc, lrr, lrc, clr, addr)
int    fieldnum;     /* a mező sorszáma */
int    ulr;          /* a mező felső sora */
int    ulc;          /* a mező bal oszlopa */
int    lrr;          /* a mező alsó sora */
int    lrc;          /* a mező jobb oszlopa */
int    clr;          /* az editálási szín */
int    *addr;        /* a mező form buffer-beli
                      kezdőcíme */

{
aputtysn(MINOSEGEK[*addr], clr, lrc-ulc+1);

```


}

Megjegyzés: A CEX2 könyvtár `selstr()` függvényének három paramétere van:

1. a string tömb NULL-val lezárva (`char **`),
2. a menü keretébe kerülő cím (`char *`) és
3. a default érték (`int`).

A rutin a CEX2 szokásos 'glcolors' színpaletta CMSG (üzenet) színét használja.

5.2.5. Betöltő függvény

A betöltő függvény alapvetően az alrekordnak a SUBFORM mezőhöz rendelt al-form mezőinek betöltésére szolgál. Ezen kívül a második paraméter értékétől függően jelzési célból hívódik.

Használat:

```
int record_load(actrec,function)
int      actrec;          /* az aktuális alrekord sorszáma */
int      function;        /* 0: betöltés, -1: jelzés,
                           1: új jelzése */
```

A SUBFORM típusú mező betöltő függvényét a `Form_Exec()` hívja meg az aktuális alrekord sorszámaival és

- 1 funkciókóddal, ha egy új alrekord definiálását kezdeményeztük,
- 0 funkciókóddal, ha a SUBFORM mezőhöz rendelt al-formba be kell tölteni az alrekordból a megfelelő értékeket,
- 1 funkciókóddal, ha a `Form_Exec()` kezelése során egy alrekordra állunk rá,
- 2 funkciókóddal, ha a `Form_Exec()` kezelése során a SUBFORM mezőre állunk rá,
- 3 funkciókóddal, ha a `Form_Exec()` kezelése során a SUBFORM mező elhagyása következik.

A betöltő függvény visszatérési értékét a futtató rendszer csak az 1 és a -3 értékű funkcióknál figyeli.

- 1 funkciókóddal a visszatérési érték 0, ha új alrekord definiálását nem engedélyezzük és nem 0, ha engedélyezzük (elsősorban limitált számú alrekord felviteléhez használható).

-3 funkciókóddal a visszatérési érték 0, ha a SUBFORM mező elhagyását nem engedélyezzük és nem 0, ha engedélyezzük.

A betöltő függvény példáját lásd a következő pontban!

5.2.6. Kimentő függvény

A kimentő függvény az aktuális alrekord értékének változása vagy új alrekord definiálása után az értékek elmentését, ill. egy alrekord törlésének engedélyezését végzi.

Használat:

```
int record_store(actrec,function)
int      actrec;          /* az aktuális alrekord sorszáma */
int      function;        /* 0: változott, 1: új rekord,
                           -1: törlés */
```

A SUBFORM típusú mező kimentő függvényét a **Form_Exec()** hívja meg az aktuális alrekord sorszámaival és

1 funkciókóddal, ha egy új alrekord definiálása történt és egy új alrekordba kell tölteni a megfelelő értékeket,

0 funkciókóddal, ha az alrekord értéke változott és az alrekordba kell tölteni a megfelelő értékeket,

-1 funkciókóddal, ha a **Form_Exec()** kezelése során egy alrekordra törlési kérés érkezett.

A kimentő függvény visszatérési értéke:

0 és 1 értékű funkcionál 0, ha az értékeket nem fogadjuk el, és nem 0, ha elfogadjuk,

-1 értékű funkcionál 0, ha a törlést nem engedélyezzük, és nem 0, ha engedélyezzük.

A következő példa egy maximálisan 20 alrekordból álló TETELEK nevű SUBFORM mező betöltő és kimentő függvénye. Az alrekordokat egy az alrekord méretének megfelelő dinamikus struktúratömb tartalmazza. A fő form "ActRec" mezője az aktuális alrekord sorszámanak, a "RecNum" mező pedig az alrekordok aktuális számának megjelenítésére szolgál.

```
#include <FORM.H>
#include <INSERT.H>      /* dinamikus struktúratömb
                           kezelése */
```



```

#include "FOFORM.FH"          /* fő form header
                                (FormEdit generálta) */
#include "TETEEK.FH"          /* SUBFORM header
                                (FormEdit generálta) */
typedef struct STETEKEK ST,*PT; /* a könnyeb írásmód
                                kedvéért */

static PT      tetelek = NULL;
static int     tetelsz = 0;
/* a Form_Fields() és a Form_Exec() hívása előtt... */
.
.
Field_Set(F_TETEKEK,(char *)&tetelsz);
Field_Set(F_RecNum ,(char *)&tetelsz);
.
.
int TETEKEK_load(actrec,function)
    int  actrec; /* az aktuális alrekord sorszáma */
    int  function; /* 0:betöltés, -1:jelzés,
                    1:új jelzése */
{
    if(function == 0) /* be kell tölteni a form
                        bufferébe */
    {
        movbytes(Form_Buffer(),tetelek+actrec,sizeof(ST));
        Field_Def_All(1);
        return(1);
    }
    if(function == 1) /* új alrekord létrehozása? */
    {
        if(tetelsz >= 20)
        {
            sherror("Maximum 20 alrekord vihető fel!",NULL);
            return(0);
        }
    }
}

```

```

    }
}

/* az alrekord sorszámának kiírása az "ActRec" mezővel */
++actrec;          /* ez 0-tól kezdődik! */
FormF_Do(Field_Set,F_Act_Rec,(char *)&actrec);
FormF_Do(Field_Put,F_Act_Rec);
return(1);
}

int TETELEK_store(actrec,function)
    int  actrec; /* az aktuális alrekord sorszáma */
    int  function; /* 0:változott, 1:új rekord,
                    -1:törlés */
{
    ST    work; /* beszúráshoz munkaterület */
    if(function == -1) /* törlés? */
    {
        /* törlés a dinamikus struktúratömbből */
        ADELETE(PT,tetelek,tetelsz,actrec);
        /* az alrekordok számát kiíratjuk a RecNum mezőn */
        FormF_Do(Field_Set, F_RecNum, (char*)&tetelsz);
        FormF_Do(Field_Put,F_RecNum,);
        return(1); /* a törlést engedélyezzük */
    }
    if(function == 1) /* új alrekord felvitele? */
    {
        /* beszúráás a dinamikus struktúratömbbe */
        AINSERT(PT,tetelek,tetelsz,actrec,work);
        /* az alrekordok számát kiíratjuk a RecNum mezőn */
        FormF_Do(Field_Set,F_RecNum,(char*)&tetelsz);
        FormF_Do(Field_Put,F_RecNum,);
    }
    movbytes(tetelek+actrec,Form_Buffer(),sizeof(ST));
    return(1);
}

```


5.2.7. Elrendelt függvény

A billentyűkhöz rendelt függvényeket a FormEdit-ben az Alt-K (Key assignment) billentyű leütése után lehet definiálni.

Ezeket a függvényeket a **Form_Exec()** run-time rutin hívja meg a megfelelő billentyű leütésére (természetesen csak akkor, ha a FormEdit-ben definiálva lett).

Használat:

```
#include <kbget.h>
#include <FORM.H>
int keyass(pressed,fieldnum,actrec)
    int    pressed;          /* a leütött billentyű kódja */
    int    fieldnum;         /* az aktuális mező sorszáma */
    int    actrec;           /* az aktuális alrekord sorszáma */
```

A paraméterek jelentése:

pressed: a leütött billentyű kódja (alsó byte ASCII, felső byte scan kód). Értelmezése a FORM.H-ban definiált SCAN() makró és a CEX (Magiszter Software) könyvtárhoz tartozó KBGET.H header-file-ban deklarált konstansokkal célszerű. Ennek segítségével hasonló funkciók egy függvényben valósíthatók meg úgy, hogy a leütött billentyű függvényében ágaztatjuk el az algoritmust.

fieldnum: annak a mezőnek a sorszáma, amelyen a futtató rendszer állt, amikor a billentyű leütése megtörtént.

actrec: ha SUBFORM-ban vagyunk, az aktuális alrekord sorszáma, egyébként -1.

Visszatérés:

0, ha a függvény visszatérése esetén nem kell csinálni semmit,
 -1, ha a függvény visszatérése után a SUBFORM mezőket
 (vagy ha SUBFORM-ban vagyunk a form összes SUBFORM
 mezőjét) újra kell írni az első alrekordtól,
 kód azaz egy értelmes billentyűkód, amelyet a futtatórendszer
 a következő leütött billentyűként értelmez.

A rutin segítségével tetszőleges funkciót végezhetünk el a billentyű leütésének hatására például help szöveg kiírása, számítások elvégzése stb.

Ha egy ilyen függvény megváltoztatja valamelyik mező értékét, akkor a függvénynek meg kell hívnia a mező újbóli kiírásához a **Field_Put()** függvényt.

Ha a függvény a teljes képernyőt elrontja (pl.: nem window-technikát használó help), akkor visszatérés előtt a függvénynek a **Form_Text()**, a **Form_Fields()** és a **Field_Select()** rutinokat meg kell hívnia (SUBFORM mező esetén a **FormF_Do()** függvényen keresztül!).

Ha az operátori kezelés során egy SUBFORM mezőn belül vagyunk, akkor az elrendelt billentyű leütésének a hatására a következő:

- ha a SUBFORM mező al-formjában az adott billentyű elrendelt billentyű, akkor az ennek megfelelő elrendelt függvény kapja meg a vezérlést,
- ha a SUBFORM mező al-formjában az adott billentyű nem elrendelt billentyű, akkor a SUBFORM mezőt tartalmazó formban lévő elrendelésnek megfelelő elrendelt függvény kapja meg a vezérlést.

A következőkben elrendelt C függvény néhány tipikus alkalmazását mutatjuk be.

A **Form_Exec()** megszakítása úgy, hogy SUBFORM esetén az aktuális alrekord éppen beütött értékeit nem kell menteni (ne hívódjon meg a SUBFORM store függvénye):

```
#include <kgget.h>    /* CEX header          */
int Break_Exec(pressed,fieldnum,actrec)
    int    pressed; /* a leütött billentyű kódja */
    int    fieldnum; /* az aktuális mező sorszáma */
    int    actrec; /* az aktuális alrekord sorszáma */
{
    Form_Break(pressed); /* A Form_Exec() visszatérési
                           értéke */
    return(ESC); /* nem kell store() hívás! */
}
```

A **Form_Exec()** megszakítása úgy, hogy SUBFORM esetén az aktuális alrekord éppen beütött értékeit menteni kell (hívódjon meg a SUBFORM store függvénye):

```
int Stop_Exec(pressed,fieldnum,actrec)
    int    pressed; /* a leütött billentyű kódja */
    int    fieldnum; /* az aktuális mező sorszáma */
    int    actrec; /* az aktuális alrekord sorszáma */
{
```



```

Form_Break(pressed); /* A Form_Exec() visszatérési
                        értéke */
return(0);           /* kell store() hívás! */
}

```

A PgUp és a PgDn billentyűk kezelése úgy, hogy ha SUBFORM-ban vagyunk, akkor a rendszer ne reagáljon rá, SUBFORM-on kívül a szokásos módon kezelje.

```

My_PgUpDown(pressed, fieldnum, actrec)

    int    pressed; /* a leütött billentyű kódja */
    int    fieldnum; /* az aktuális mező sorszáma */
    int    actrec;   /* az aktuális alrekord sorszáma */
{
    if(actrec < 0) /* nem SUBFORM-ban vagyunk? */
        return(pressed);
    return(0);    /* PgUp, PgDn letiltva */
}

```

SUBFORM esetén, ha az utolsó alrekordon vagyunk, akkor a DOWN billentyű kezelődjön ^A-ként, tehát automatikus Append funkcióként. Az alrekordok számát a *recnum* változóban tároljuk.

```

#include <kbget.h> /* CEX header */

My_Down(pressed, fieldnum, actrec)

    int    pressed; /* a leütött billentyű kódja */
    int    fieldnum; /* az aktuális mező sorszáma */
    int    actrec;   /* az aktuális alrekord sorszáma */
{
    if(actrec < 0) /* nem SUBFORM-ban vagyunk? */
        return(pressed);
    if(actrec <= recnum-1) /* nem az utolsó alrekordon
                           állunk? */
        return(pressed);
    return(CTR_A); /* A DOWN legyen ^A */
}

```


5.3. Keretprogram

Az M-FORM rendszer használatának végső célja mindig valamilyen alkalmazói program írása. Az M-FORM rendszer a tervezést és a programozást nagymértékben megkönnyíti. A rendszer alkalmazásához csupán azt a programot kell megírni, amely a nem az M-FORM hatáskörébe tartozó feladatokat látja el (file-kezelés, környezet kezelése, felhasználói beépített függvények, help stb.). Ezek munkaigényessége a kívánt rendszer összetettségétől függ. A legegyszerűbb esetekben ez standard módon megoldható, mint azt a következő fejezetben láthatjuk.

A következő séma egy egyszerű adatbeviteli metódus vázlata, amelyben csak hozzáfűzni és módosítani lehet.

1. form beolvasása ->**Form_Read()**
2. file-megnyitás (ha még nincs, akkor létrehozás)
3. képernyő inicializálás ->**Form_Text()**
4. feldolgozó ciklus

pozicionálás az aktuális rekordra

form bufferének inicializálása ->**Form_Init()**

rekord beolvasása a felhasználó bufferébe

ha van rekord (nem új), akkor form feltöltése ->**Field_Set()**

a mezők kiírása a képernyőre ->**Form_Fields()**

a rekord kezelése ->**Form_Exec()**

ha volt változás ->**Form_Changed()**

visszatöltés a felhasználói bufferbe ->**Field_Get()**

pozicionálás az aktuális rekordra

a rekord visszaírása

pozicionálás állítása a **Form_Exec()** visszatérésétől függően

5. file-lezárások

6. a form lezárása ->**Form_Close()**

5.4. Egyszerű példaprogram

A következő C forrásprogram az előző vázlat C változata. A program alkalmas bármilyen FormEdit-tel létrehozott form alapján rekordok felvitelére és módosítására. Ebben az esetben természetesen a form nem hivatkozhat felhasználói C függvényekre, tehát nem lehet USER és SUBFORM típusú mezője, egyik mezőhöz sem lehet beolvasó vagy ellenőrző függvényt rendelni és nem lehet a billentyűkhöz sem függvényt rendelni.

A létrehozott file-ban a rekord hossza a form bufferének hossza lesz, azaz a form I/O mezőinek összesített hossza.

A program új rekordot csak az állomány végehez képes hozzáfűzni. A beolvasandó form és a kezelendő adatfile azonos néven (.FTX és .FDB típussal) azonos könyvtárban kell hogy legyen.

Ha a programnak nem adjuk meg a form és egyben az adatfile nevét, akkor saját nevéből képzi azt. Tehát ha a program neve PELDA.EXE és híváskor nem adunk paramétert, akkor a program az azonos könyvtárban lévő PELDA.FTX form-file-t és a PELDA.FDB adatfile-t feltételezi.

```
#include <string.h> /* standard */
#include <fcntl.h> /* standard */
#include <stdio.h> /* standard */
#include <io.h> /* standard */
#ifdef TURBOC
    #include <alloc.h> /* standard TurboC */
#else
    #include <malloc.h> /* standard Microsoft */
#endif
#include <kbget.h> /* CEX */
#include <setattr.h> /* CEX */
#include <cex2.h> /* CEX2 */
#include <form.h> /* MI-FORM */
int    changed; /* a rekord változott? */
main(argv)
```



```
int  argc;
char *argv[];
{
char  filename[80]; /* a form és az adatfile neve */
char  *p; /* a 'filename' állításához */
char  *buffer; /* a felhasználói buffer */
int  buflen; /* a buffer hossza */
int  nfields; /* a mezők száma */
int  handle; /* a file azonosítója */
int  recnum; /* a rekordok száma */
int  actrec; /* az aktuális rekord sorszáma */
long  pos; /* byte-pozíció a file-ban */
int  offset; /* a mező offset-je */
int  fieldnum; /* aktuális mező */
int  retval; /* a Form_Exec() visszatérési értéke */
if (argc > 1)
    p = argv[1]; /* file neve megadva */
else
    p = argv[0]; /* a program nevével megegyező név */
strcpy(filename,p);
p = strchr(filename,'.');
if (p != NULL)
    *p = '\0'; /* file típus törlése */
if (Form_Read(filename) < 0)
{
    fprintf(stderr,"A \"%s\" form nem olvasható be!\n");
    exit(-1);
}
strcat(filename,".FDB"); /* a file neve */
handle = open(filename,O_RDWR | O_BINARY); /* nyitás read/write */
if (handle < 0) /* ha még nem létezett: létrehozás */
    handle = open(filename,O_CREAT | O_RDWR | O_BINARY);
    buflen = Form_Size(); /* a form/felhasználói
```



```

        buffer hossza */
pos = filelength(handle); /* a file hossza */
if (pos % (long) bufflen != 0) /* nem egész számú
        többszörös? */
{
    fprintf(stderr, "A rekordhossz hibás a \"%s\",
        "file-ban!\n", filename);
    exit(-1);
}
recnum = (int)(pos / (long)bufflen); /* a rekordok száma */
actrec = 0;
buffer = malloc(bufflen); /* felhasználói buffer */
if (buffer == NULL)
{
    fprintf(stderr, "Kevés a memória!\n");
    exit(-1);
}
Form_Wsave(); /* képernyő mentése */
Form_Text(); /* konstans képernyőrész kiírása */
nfields = Field_Number(); /* a mezők száma */
NewRecord: /* a ciklikus feldolgozás */
Form_Init(); /* form-buffer inicializálás */
pos = actrec * bufflen; /* pozíció a file-ban */
if (lseek(handle, pos, 0) >= 0) /* ráállás a rekordra
        és beolvasás */
{
    if (read(handle, buffer, bufflen) == bufflen)
    {
        for (fieldnum = 0; fieldnum < nfields; ++fieldnum)
        {
            offset = Field_Offset(fieldnum);
            Field_Set(fieldnum, buffer+offset);
        }
    }
}

```

```

    }
}

Form_Fields(); /* mezők értékének kiírása */
changed = 0; /* még nem volt változás */
ExecAgain:
retval = Form_Exec(0); /* a form-"végrehajtása" */
fieldnum = Field_No_Valued(); /* első nem kitöltött mező */
if (fieldnum >= 0)
{
    if (actrec >= recnum) /* új rekord? */
    {
        shmsg("Rekord nincs kitöltve, nincs felvitel.", NULL);
        if (actrec == 0)
            retval = ESC; /* első rekordnál: kiszállunk */
        --actrec;
        goto NewRecord; /* visszalépünk */
    }
    shmsg("A", Field_Name(fieldnum), "nincs kitöltve!", NULL);
    Field_Select(fieldnum);
    goto ExecAgain;
}

if (changed) /* a Form_Changed() állíthatta */
{
    /* a változott rekord kimentése */
    for (fieldnum = 0; fieldnum < nfields; ++fieldnum)
    {
        offset = Field_Offset(fieldnum);
        Field_Get(fieldnum, buffer+offset);
    }
    lseek(handle, pos, 0); /* aktuális rekordra */
    write(handle, buffer, buflen); /* visszairás */
    if (actrec >= recnum)
        ++recnum; /* új rekord volt */
}

```



```

switch (retval)
{
    case ESC:
        free(buffer); /* buffer felszabadítása */
        close(handle); /* adat-file lezárása */
        Form_Wclose(); /* képernyő visszaállítása */
        Form_Close(); /* a form felszabadítása */
        exit(0);

    case SCAN(PGUP): /* előző rekord */
        if (actrec > 0)
            --actrec;
        break;

    case SCAN(PGDN): /* következő rekord */
        ++actrec;
        break;
}

goto NewRecord;
}

Form_Changed(fieldnum, actrec)
/* A RunTime hívja, ha tényleges változás történt a form
   bufferében */
int fieldnum;
int actrec;
{
    changed = 1; /* csak ilyenkor fogjuk a rekordot visszaírni */
}

```

Megjegyzések:

- A programban a felhasználói buffer helyett használhattuk volna a form bufferét is, hiszen a felhasználói és a form buffer tartalma között nincs különbség. Ekkor a **Field_Set()** ciklikus hívása helyett direkt módon a form bufferébe olvashatunk, és olvasás után **aField_Def_All(1)**; hívással jelezhetjük a futtatórendszernek, hogy minden mező fel van töltve. Természetesen kiíraskor sem kell a **Field_Get()** függvényt ciklikusan hívni, a form bufferét írhatjuk ki.

- A program generálását a fenti forrásból (SAMPLE.C) a következő parancsokkal végezhetjük (feltételezve a nyelvi környezetnek megfelelő konfigurációt, pl. Microsoft-nál a LIB= Environment, ill. TurboC-nél a TURBOC.CFG meglétét):

Microsoft C:

```
>CL sample.c -link sform scex2 scex
```

Turbo C:

```
>TCC -DTURBOC sample.c forms.lib cex2s.lib cexs.lib
```

vagy Integrált TC esetén a PRJ file tartalma:

```
sample.c,  
forms.lib,  
cex2s.lib,  
cexs.lib.
```

5.5. Teljes példa

A következőkben bemutatott példa szinte teljeskörűen kihasználja az M-FORM lehetőségeit. Mivel egy működő, önmagában is hasznos programmal kívántuk az M-FORM használatát demonstrálni, ezért a program tartalmaz nem az M-FORM magyarázatához tartozó funkciókat is, de reméljük ezek tanulmányozása sem bizonyul elvesztegetett időnek.

5.5.1. A program specifikációja

Az ISM program egy cím-, és telefonjegyzéket kezelő program, amely a szokásos funkciókon kívül név- és születésnap, valamint kölcsönadott és kölcsönvett tárgyak regisztrálására is alkalmas.

A program adatbázisát egy szöveg-file alkotja, amely kötött formában tartalmazza az egyes ismerősök vagy közületek adatait. Az adatbázis indításkor teljes egészében beolvasásra kerül, és a kiszállásig nincs file művelet. A program kimentést csak külön kérésre és csak változtatás esetén végez.

Definiált funkciók:

- Felvitel
- Módosítás
- Törlés
- Lista generálása
- Keresés

- Lekérdezés
- Szűrő beállítása
- Név és születésnapok lekérdezése
- Névnapi beállítása táblázat segítségével

A program használja az ISM.NDX file-t, amely az egyes gyakoribb névnapok dátumát tartalmazza (csak input file, editorral bővíthető).

5.5.2. A programhoz tartozó form

ISM

A form neve: ISM A form helye: 0,0 - 24,79 Leírás: Az ISM program formja

1. oldal

Címek és telefonszámok	
Név: _____	Szűrő: _____
Személyi adatok	
Cím: _____	
Tel.: _____	
Szül.: _____ Szem. sz. _ - _ - _	Névnap _____
Hivatali információk	
Cím: _____	
Tel1: _____	Tel2: _____
Tartozik	
Követel	
Megj.: _____	
Segítség: F1	

0 % Sorszám: _____ Címek száma: _____ Nem volt változás 100 %

1. mező: name (input)

Mezőleírás : Név
Státusz sor : Adja meg az azonosító nevet!
Típus : Text
Helye : 1:8 Hossza: 60
Formátum : "L"
Ellenőrző függv.: "convstring"

2. mező: kind (input)

Mezőleírás : Szűrő-érték
Státusz sor : Teljes info: 0 Személyes: 1 Közület: 2
Típus : unsigned Byte
Helye : 1:77 Hossza: 1
Beolvasó függv. : "kind-input"

3. mező: addr (input)

Mezőleírás : Otthoni cím
Státusz sor : Adja meg az otthoni címet!
Típus : Text
Helye : 3:8 Hossza: 71
Ellenőrző függv.: "convstring"

4. mező: tel (input)

Mezőleírás : Otthoni telefonszám
Státusz sor : Adja meg az otthoni telefonszámot!
Típus : Text
Helye : 4:8 Hossza: 30

5. mező: birth (input)

Mezőleírás : Születési dátum
Státusz sor : Adja meg a születési dátumot!
Típus : Date (dátum forma: Magyar)
Helye : 5:8 Hossza: 8
Ellenőrző függv.: "birth-update"

6. mező: code (input)

Mezőleírás : Személyi szám

Státusz sor : Adja meg a személyi számot!
Típus : Text
Helye : 5:39 Hossza: 13
Formátum : "#####-####"
Ellenőrző függv.: "codeupdate"

7. mező: ndmonth (input)

Mezőleírás : Névnep hónapja
Státusz sor : Adja meg a névnep hónapját vagy használja az ALT-N-t!
Típus : Byte
Helye : 5:74 Hossza: 2
Ellenőrző függv.: "ndmonthupdate"

8. mező: ndday (input)

Mezőleírás : Névnep napja
Státusz sor : Adja meg a névnep napját vagy használja az ALT-N-t!
Típus : Byte
Helye : 5:77 Hossza: 2
Ellenőrző függv.: "nddayupdate"

9. mező: office (input)

Mezőleírás : A hivatal neve
Státusz sor : Adja meg a hivatal nevét!
Típus : Text
Helye : 7:1 Hossza: 78
Ellenőrző függv.: "convstring"

10. mező: oaddr (input)

Mezőleírás : A hivatal címe
Státusz sor : Adja meg a hivatal címét!
Típus : Text
Helye : 8:7 Hossza: 72
Ellenőrző függv.: "convstring"

11. mező: otel1 (input)

Mezőleírás : Az első hivatali telefonszám
Státusz sor : Adja meg a hivatali telefonszámot!

Típus : Text
 Helye : 9:7 Hossza: 30

12. mező: otel2 (input)

Mezőleírás : A második hivatali telefonszám
 Státusz sor : Adja meg a hivatali telefonszámot!
 Típus : Text
 Helye : 9:49 Hossza: 30

13. mező: debts (input)

Mezőleírás : Tartozások
 Státusz sor :
 Típus : Subform
 Helye : 11:1 - 13:78 3 rekord kiírásához elég.
 Betöltő függvény: "debtsload"
 Kimentő függvény: "debtsstore"

14. mező: claims (input)

Mezőleírás : Követelések
 Státusz sor :
 Típus : Subform
 Helye : 15:1 - 17:78 3 rekord kiírásához elég.
 Betöltő függvény: "claimsload"
 Kimentő függvény: "claimsstore"

15. mező: note (input)

Mezőleírás : Megjegyzés
 Státusz sor : Ha van megjegyzése: írja be!
 Típus : Text
 Helye : 19:8 Hossza: 71
 Ellenőrző függv.: "convstring"

16. mező: StatLine (output)

Mezőleírás : Százalékos pozíció az adatállományban
 Státusz sor :
 Típus : Text
 Helye : 23:1 Hossza: 78

17. mező: ActRec (output)

Mezőleírás : Az aktuális rekord sorszáma
 Státusz sor :
 Típus : unsigned Int
 Helye : 24:18 Hossza: 4

18. mező: RecNum (output)

Mezőleírás : A rekordok száma
 Státusz sor :
 Típus : unsigned Int
 Helye : 24:44 Hossza: 4

19. mező: Changed (output)

Mezőleírás : Változás jelzése
 Státusz sor :
 Típus : Text
 Helye : 24:53 Hossza: 17
 Alapérték : "Nem volt változás"

Billentyű elrendelés:

Elrendelt bil.	C függvény
F1	Help
	Search
Ctrl-F	FreeMem
Ctrl-PgUp	FirstRec
Ctrl-PgDn	LastRec
Alt-E	Break
Alt-K	Query
Alt-L	Break
Alt-M	Break
Alt-N	NameDay
Alt-S	Filter
Alt-T	Delete
Alt-U	Insert

debts

A form neve: debts A form helye: 11,3 - 11,78 A ISM form al-formja

Leírás: Egy tartozást kezelő al-form

1. oldal

--

1. mező: debt (input)

Mezőleírás : Egy tartozás

Státusz sor : Tartozások. Beszúrás: INS Hozzáadás: ^A Törlés: DEL

Típus : Text

Helye : 11:3 Hossza: 76

Formátum : "n"

Ellenőrző függv.: "convstring"

claims

A form neve: claims A form helye: 15,3 - 15,78 A ISM form al-formja

Leírás: Egy követelést kezelő al-form

1. oldal

--

1. mező: claim (input)

Mezőleírás : Egy követelés

Státusz sor : Követelés. Beszúrás: INS Hozzáadás: ^A Törlés: DEL

Típus : Text

Helye : 15:3 Hossza: 76

Formátum : "n"

Ellenőrző függv.: "convstring"

5.5.3. A példaprogram forrása

```
/* ISM.DAT: konstansok, típusok és változók */
#include <stdio.h> /* standard C header */
#include <string.h> /* standard C header */
/* CEX2 header (Magiszter Software) */
#include <cex2.h> /* általános definíciók */
#include <ainsert.h> /* dinamikus tömbkezelés */
/* CEX header (Magiszter Software) */
#include <kbget.h> /* billentyűkódok */
#include <setattr.h> /* színek */
#include <form.h> /* M-FORM header */
#include "ISM.FH" /* a FormEdit által generált header */
#ifdef LINTARGS
    #include "ISM.LNT" /* LINT ellenőrzéshez */
#endif
#define MAXBUF    256 /* az I/O buffer mérete */
#define NO        0
#define YES       1
/* Egy rekord struktúrája (az alapja: az SISIM struktúra,
** amely az ISM.FH header-file-ban van definiálva)
*/
typedef struct ISM {
    char    *name; /* a név */
    unsigned char    kind; /* a rekord milyensége */
    char    *addr; /* az otthoni cím */
    char    *tel; /* az otthoni telefonszám */
    long    birth; /* a születési dátum */
    char    *code; /* a személyi szám */
    unsigned char    ndmonth; /* a névnap hónapja */
    unsigned char    ndday; /* a névnap napja */
    char    *office; /* a munkahely vagy hivatal */
    char    *oaddr; /* a munkahely címe */
}
```



```

char *otel1; /* a munkahely első telefonszáma */
char *otel2; /* a munkahely második telefonszáma */
int  debtn; /* a tartozások száma */
char **debts; /* a tartozások dinamikus stringtömbje */
int  claimn; /* a követelések száma */
char **claims; /* a követelések dinamikus stringtömbje */
char *note; /* megjegyzés */

/* char StatLine[78];      csak output mező */
/* int ActRec;             csak output mező */
/* int RecNum;             csak output mező */
/* char Changed[17];       csak output mező */
    } SISM, *PISM;
/* a hónapok stringjei */
/*gl*/ char *months[] =
    {"XXX", /* nincs 0-dik! */
     "Jan", "Feb", "Már", "Ápr",
     "Máj", "Jún", "Júl", "Aug",
     "Sep", "Okt", "Nov", "Dec"
    };
/* Egy rekord lehetséges szűrő-értékei */
#define MIND      0 /* személyes és hivatali */
#define SZEMELYES 1 /* személyes */
#define KOZULET   2 /* közület */
/* A kiválasztási szűrő szövegei: */
/*gl*/ char *FILTER[] = {
    "Mind",
    "Személyes",
    "Közület",
    0};
/* globális adatok */
/*gl*/ PISM isms = NULL; /* adatbázis a memóriában */
/*gl*/ int ismn = 0; /* a rekordok száma */
/*gl*/ PISM actpi; /* az aktuális rekord mutatója */

```

```

/*gl*/ int actism; /* az aktuális rekord sorszáma */
/*gl*/ int filter = MIND; /* az aktuális szűrő */
/*gl*/ int changed = NO; /* volt változás? */
/*gl*/ int onlyanno= NO; /* csak születés/névnap
                                lekérdezése? */
/*gl*/ char buffer[MAXBUF]; /* globális munka buffer */
/*gl*/ char insbuffer[80]; /* buffer egy új rekord
                                beszúrásához */
/*gl*/ char glcolors[CHHELP+1]; /* globális színek a
                                CEX2-höz */
extern int errno; /* standard C változó */

```



```

/* ISM.C
**
** Ismerősök, címek és telefonszámok regisztrálása
**
** A program hívása:
**
** >ISM [-b] [-a]
**
** ahol  -b a fekete/fehér üzemmódot jelöli
**        -a csak az évfordulók kiírását kérjük
**/
#include "ISM.DAT" /* globális adatok */
#include "ISM.FIC" /* FORMEDIT által generált
                    form-létrehozó forrás*/
#include "ISM.FCT" /* FORMEDIT által generált
                    C függvénytábla */
static char  FILENAME[] = "ISM"; /* a FORM és
                                   adatfile neve */
static char  bwcolors[CHHELP+1] = {
                                   /* fekete/fehér színek */
    GRAY,
    WHITE,
    COLOR(DGRAY,GRAY),
    COLOR(BLACK,GRAY),
    COLOR(WHITE,GRAY)
};

void main(argc,argv)
int  argc;
char  **argv;
{
int  formnum; /* a form sorszáma */
int  pressed = 0; /* a leütött billentyű kódja */
int  is_bw = NO; /* fekete/fehér üzemmód? */

```

```

/* program-paraméterek elemzése */
while (++argv)
{
    if (strcmp(*argv, "-b") == 0) /* fekete/fehér üzemmód */
        is_bw = YES;
    else if (strcmp(*argv, "-a") == 0) /* csak évfordulók */
    {
        pressed = SCAN(ALT_E); /* az előre ütött bill. */
        onlyanno = YES; /* globális jelző */
    }
}

if ((formnum = ISM_crea()) < 0) /* a form dinamikus
                                létrehozása */
{
    sherror("Az ISM form nem hozható létre!", NULL);
    exit(4);
}

Form_Select(formnum); /* kiválasztjuk a form-ot */
Set_Form_Funcs(ISM_FCT); /* a C függvénytábla illesztése */
if (is_bw) /* fekete/fehér üzemmód? */
{
    Form_Colors(1, bwcolors); /* a form-színek beállítása */
    movbytes(glcolors, bwcolors, 5); /* a CEX2 színek
                                    beállítása */
    /* az al-form-ok színbeállítása */
    formnum = Search_Form(Field_Name(F_debts));
    Form_Do(formnum, Form_Colors, 1, bwcolors);
    formnum = Search_Form(Field_Name(F_claims));
    Form_Do(formnum, Form_Colors, 1, bwcolors);
}
else
    Form_Colors(0, glcolors); /* a form színei a CEX2 színekbe */
Form_Wsave(); /* a képernyő lementése */

```



```

Form_Clear(); /* a képernyő törlése */
Form_Text(); /* a konstans képernyőrészek */
Form_Init(); /* a mezők kiírása előtt */
Form_Fields(); /* a beolvasás alatt is kell */
if (readdatas(FILENAME) == NO) /* adatok beolvasása */
    endism(4); /* hibás beolvasás */
if (ismn == 0) /* ha még nincs rekord, akkor */
{
    if (onlyanno) /* évfordulók nincsenek */
        endism(3);
    pressed = SCAN(ALT_U); /* új felvitelével kezdjük */
}
process(pressed); /* az adatfeldolgozás */
endism(0); /* vége */
}

static endism(code)
/* A program befejezése a képernyő visszaállításával */
int code; /* a program kiszállási kódja */
{
    Form_Wclose(); /* a képernyő visszaállítása */
    Form_Close(); /* a form lezárása */
    exit(code);
}

static process(pressed)
/* Az adatfeldolgozás vezérlése */
int pressed; /* az előre leütött billentyű */
{
    int oldrec; /* az előző rekord sorszáma */
    int retval; /* a Form_Exec() visszatérése */
    SISM si; /* beszúráshoz */
    actism = -1; /* az első rekord kereséséhez */
    nextrecord(); /* az első rekord keresése */
    NewRecord: /* lecsurgó ciklus */

```

```
Form_Init(); /* a form mezők inicializálása */
if (actism >= 0) /* ha van rekord... */
{
    formload(); /* betöltés az adatbázisból */
    Form_Fields(); /* kiíratás a képernyőre */
    statline(actism+1); /* százalékos kijelzés */
}
oldrec = actism; /* elmentjük a formstore()-nak */
OnceAgain: /* belső lecsurgó ciklus */
retval = Form_Exec(pressed); /* az operátoré a szó */
pressed = 0; /* már nem kell */
if (oldrec >= 0)
    formstore(oldrec); /* az adatok kimentése */
switch (retval)
{
    case ENTER: /* új rekord kell */
        goto NewRecord;
    case ESC:
        if (!changed) /* nem volt változás? */
            return;
        if (shgetans("Biztos, hogy ki akar szállni",
                    "a megváltozott adatok lementése nélkül?",
                    NULL))
            return;
        goto OnceAgain;
    case SCAN(PGUP): /* előző rekord */
        if (prevrecord())
            goto NewRecord;
        goto OnceAgain;
    case SCAN(PGDN): /* következő rekord */
        if (nextrecord())
            goto NewRecord;
        goto OnceAgain;
```



```

case SCAN(ALT_E):
    Anno();
    if (onlyanno) /* csak az évfordulók kellettek */
        return;
    goto OnceAgain;
case SCAN(ALT_L):
    Report();
    goto OnceAgain;
case SCAN(ALT_M): /* az adatok kimentése */
    if (!changed)
    {
        shmsg( "Az adatok nem változtak!",
               "A programból az ESC-pel tud kiszállni!", NULL);
        goto OnceAgain;
    }
    writedatas(FILENAME);
    changed = NO;
    goto NewRecord;
case SCAN(ALT_U): /* új rekord beszúrása */
    changed = YES;
    statclose(7); /* százalékos kijelzés lezárása */
    actism = findplace(insbuffer);
    if (actism < ismn)
    {
        if (sacompere(insbuffer, (isms+actism)->name) == 0)
            goto NewRecord;
    }
    memset((char *)&si, 0, sizeof(SISM));
    si.name = newname(insbuffer);
    statclose(7);
    AINSERT(PISM, isms, ismn, actism, &si);
    statinit(ismn, Field_Length(F_StatLine),
            glcolors[CEDIT],

```

```

        Field_Row(F_StatLine),
        Field_Col(F_StatLine));

    goto NewRecord;

case SCAN(ALT_T): /* az aktuális törlése */
    changed = YES;
    statclose(7); /* százalékos kijelzés lezárása */
    freeism(actism); /* a rekordelem felszabadítása */
    ADELETE(PISM,isms,ismn,actism);
    if (actism >= ismn)
        --actism;
    statinit(ismn,Field_Length(F_StatLine),
            glcolors[CEDIT],
            Field_Row(F_StatLine),
            Field_Col(F_StatLine));

    if (!inselected(actism))
        { /* pozícionálás */
            if (nextrecord() == NO)
                {
                    if (prevrecord() == NO)
                        {
                            filter = MIND;
                            prevrecord();
                        }
                }
        }

    goto NewRecord;

default:
    sherror(convnum(retval >> 8),NULL);
}

}

static formload()
/* Feltölti a form bufferét az adatbázisból */
{

```



```

register PISM pi; /* a gyorsabb kezelés végett */
int val; /* a Field_Set()-nek cím kell */
actpi = pi = isms + actism; /* megjegyezzük az aktuálist */
/* C stringekhez a 'setstring()' rutint használjuk */
setstring(F_name , pi->name );
Field_Set(F_kind , (char*)&pi->kind );
setstring(F_addr , pi->addr );
setstring(F_tel , pi->tel );
if (pi->birth)
    Field_Set(F_birth , (char*)&pi->birth );
setstring(F_code , pi->code );
if (pi->ndmonth)
{
    Field_Set(F_ndmonth , &pi->ndmonth );
    Field_Set(F_ndday , &pi->ndday );
}
setstring(F_office , pi->office );
setstring(F_oaddr , pi->oaddr );
setstring(F_otel1 , pi->otel1 );
setstring(F_otel2 , pi->otel2 );
Field_Set(F_debts , (char*)&pi->debtn );
Field_Set(F_claims , (char*)&pi->claimn );
setstring(F_note , pi->note );
val = actism + 1;
Field_Set(F_ActRec, (char *)&val); /* az aktuális rekord
                                     sorszáma */
Field_Set(F_RecNum, (char *)&ismn); /* a rekordok száma */
if (changed)
    Field_Set(F_Changed, "Volt változás");
set_protect(); /* védetség beállítása */
}
static setstring(fieldnum, from)
/* Egy TEXT mező értékét állítja be a megfelelő

```

```
C string alapján */
int  fieldnum; /* a mező sorszáma */
char *from; /* a C string */
{
if (from == NULL) /* nincs érték */
    return;
strtotxt(from,buffer,Field_Length(fieldnum));
/* konvertálás */
Field_Set(fieldnum,buffer); /* az érték beállítása */
}
static formstore(rec)
/* A form bufferéből az adatbázisba való mentés */
int  rec;
{
register PISM  pi; /* a gyorsabb kezelés végett */
pi = isms + rec;
/* A C stringekhez a 'getstring(); rutint használjuk */
getstring(F_name      ,  &(pi->name)  );
Field_Get(F_kind      ,  &(pi->kind)  );
getstring(F_addr      ,  &(pi->addr)  );
getstring(F_tel       ,  &(pi->tel)   );
if (Field_Valued(F_birth))
    Field_Get(F_birth  , (char*)&pi->birth );
else
    pi->birth = 01;
getstring(F_code      ,  &(pi->code)  );
if (Field_Valued(F_ndmonth))
{
    Field_Get(F_ndmonth ,  &pi->ndmonth );
    Field_Get(F_ndday   ,  &pi->ndday   );
}
else
{
```



```

    pi->ndmonth = 0;
    pi->ndday = 0;
}

getstring(F_office , &(pi->office) );
getstring(F_oaddr , &(pi->oaddr) );
getstring(F_otel1 , &(pi->otel1) );
getstring(F_otel2 , &(pi->otel2) );
getstring(F_note , &(pi->note) );
}

static freeism(act)
/* Felszabadítja az adott rekord által lefoglalt memóriát */
int act;
{
    register PISM pi;
    pi = isms+act;
    freestr(pi->name);
    freestr(pi->addr);
    freestr(pi->tel);
    freestr(pi->code);
    freestr(pi->office);
    freestr(pi->oaddr);
    freestr(pi->otel1);
    freestr(pi->otel2);
    freestr(pi->note);
    if (pi->debtn)
        free(pi->debts);
    if (pi->claimn)
        free(pi->claims);
}

static freestr(str)
/* Egy allokált string által lefoglalt
    memória felszabadítása */
char *str;

```

```
{
if (str)
    free(str);
}

static getstring(fieldnum,ptr)
/* Egy TEXT mező értékének beszédése allokált C stringbe */
int fieldnum; /* a mező sorszáma */
char **ptr; /* az eredeti string címe */
{
if (Field_Valued(fieldnum) == 0) /* nincs érték? */
{
if (*ptr) /* volt érték? */
{
free(*ptr);
*ptr = NULL;
}
return; /* nincs csere? */
}

txttostr(Form_Buffer()+Field_Offset(fieldnum),
          Field_Size(fieldnum),buffer);

if (*ptr == NULL || strcmp(*ptr,buffer)) /* van változás? */
    *ptr = xchgstr(buffer,*ptr); /* cseréljük az értéket */
}

Form_Changed(fieldnum,actrec)
/* A FORM.LIB függvényt kiváltó, változáskezelő */
{
if (!changed)
{
changed = YES;
Form_Do(0,Field_Set,F_Changed,"Volt változás!");
Form_Do(0,Field_Put,F_Changed);
}
}
```



```

/* ----- Input - Output rutinok ----- */
#define ENDLINES (NAME-1)
#define NAME 0
#define KIND (NAME +1)
#define ADDR (KIND +1)
#define TEL (ADDR +1)
#define BIRTH (TEL +1)
#define CODE (BIRTH +1)
#define NAMEDAY (CODE +1)
#define OFFICE (NAMEDAY+1)
#define OADDR (OFFICE +1)
#define OTEL1 (OADDR +1)
#define OTEL2 (OTEL1 +1)
#define DEPTS (OTEL2 +1)
#define CLAIMS (DEPTS +1)
#define NOTE (CLAIMS +1)
static char *KW[] = {
    "NAME=",
    "KIND=",
    "ADDR=",
    "TEL=",
    "BIRTH=",
    "CODE=",
    "NAMEDAY=",
    "OFFICE=",
    "OADDR=",
    "OTEL1=",
    "OTEL2=",
    "DEPTS=",
    "CLAIMS=",
    "NOTE=",
    NULL
};

```

```
static FILE *fp;
static char *getline();
static int readdatas(fname)
/* Beolvassa az adatokat a memóriába */
char *fname; /* a file neve */
{
    char filename[80]; /* a file-típus kezeléséhez */
    int i; /* ciklusváltozó */
    register PISM pi; /* a gyorsabb kezelés végett */
    int dummy; /* a 'getline()' paramétere */
    strcpy(filename, fname);
    setext(filename, "FIL"); /* a file ".FIL" típusú */
    if ( (fp = fopen(filename, "r")) == NULL)
        return(YES); /* még nincs file, OK */
    fgets(buffer, MAXBUF, fp); /* első sor: */
    ismn = i = atoi(buffer); /* a meglévő rekordok száma */
    if (i <= 0)
    {
        fclose(fp);
        return(YES);
    }
    statinit(ismn, Field_Length(F_StatLine),
             /* hol tartunk a beolvasásban */
             glcolors[CEDIT],
             Field_Row(F_StatLine),
             Field_Col(F_StatLine));
    Field_Set(F_RecNum, (char *)&ismn);
    Field_Put(F_RecNum);
    isms = (PISM) myalloc(i * sizeof(SISM)); /* a belső adatbázis */
    pi = isms;
    memset((char *)pi, 0, i * sizeof(SISM));
    getline(&dummy);
    for (i = 1; i <= ismn; ++i)
```



```

{
    statline(i); /* itt tartunk a beolvasásban */
    Field_Set(F_ActRec, (char *)&i);
    Field_Put(F_ActRec);
    rddata(pi++); /* a rekord beolvasása */
}

if (ferror(fp)) /* volt olvasási hiba? */
{
    sherror(filename, NULL);
    fclose(fp);
    return(NO);
}

fclose(fp);
return(YES);
}

static rddata(pi)
/* Egy rekord beolvasása */
register PISM pi; /* az aktuális rekord mutatója */
{
    int linetyp; /* a beolvasott sor típusa */
    register char *p; /* munkaváltozó */
    int i; /* a stringtömb mérete */
    char *s; /* az AINSERT-hez */
    if (memcmp(buffer, KW[NAME], 5) != 0) /* nem név-sor? */
    {
        sherror("Hibás sor", buffer, NULL);
        endism(4);
    }
    pi->name = newname(buffer+5);
    p = getline(&linetyp);
    while (linetyp > NAME)
    {
        switch (linetyp)

```

```
{  
    case ENDLINES: /* nincs több sor */  
        break;  
    case KIND:  
        pi->kind = atoi(p);  
        break;  
    case ADDR:  
        pi->addr = newname(p);  
        break;  
    case TEL:  
        pi->tel = newname(p);  
        break;  
    case BIRTH:  
        pi->birth = atodat(p,1);  
        break;  
    case CODE:  
        pi->code = newname(p);  
        break;  
    case NAMEDAY:  
        pi->ndmonth = atoi(p);  
        p = strchr(p, '/') + 1;  
        pi->ndday = atoi(p);  
        break;  
    case OFFICE:  
        pi->office = newname(p);  
        break;  
    case OADDR:  
        pi->oaddr = newname(p);  
        break;  
    case OTEL1:  
        pi->otel1 = newname(p);  
        break;  
    case OTEL2:
```



```

        pi->otel2 = newname(p);
        break;
    case DEPTS:
        i = pi->debtn;
        s = newname(p);
        AINSERT(char**,pi->debts,pi->debtn,i,&s);
        break;
    case CLAIMS:
        i = pi->claimn;
        s = newname(p);
        AINSERT(char**,pi->claims,pi->claimn,i,&s);
        break;
    case NOTE:
        pi->note = newname(p);
        break;
    }
    p = getline(&linetyp);
}

static writedatas(fname)
/* A belső adatbázis kiírása file-ba */
char *fname; /* a file neve */
{
    char filename[80]; /* a file-típus kezeléséhez */
    int i; /* ciklusváltozó */
    register PISM pi; /* az aktuális rekord mutatója */
    strcpy(filename,fname);
    setfext(filename,"FIL"); /* a file ".FIL" típusú */
    makebakfile(filename); /* mentő file képzése */
    if ( (fp = fopen(filename,"w")) == NULL)
    {
        sherror(filename);
        return(YES);
    }
}

```

```

    }

shres("Folyik az adatok kiírása...",filename,NULL);
fprintf(fp,"% d\n",i = ismn); /* a rekordok száma */
pi = isms;
for (i = 1; i <= ismn; ++i)
{
    statline(i); /* itt tartunk a kiírásban */
    Field_Set(F_ActRec,(char *)&i);
    Field_Put(F_ActRec);
    wrdata(pi++);
}

shres(NULL);
if (ferror(fp)) /* volt írási hiba? */
{
    sherror(filename,NULL);
    fclose(fp);
    unlink(filename);
    return(NO);
}

fclose(fp);
return(YES);
}

static wrdata(pi)
/* Egy rekord kiírása */
register PISM pi; /* az aktuális rekord mutatója */
{
    register int i; /* dinamikus stringtömbhöz */
    putstring(NAME ,pi->name );
    buffer[0] = pi->kind + '0';
    buffer[1] = '\\0';
    putstring(KIND ,buffer );
    putstring(ADDR ,pi->addr );
    putstring(TEL ,pi->tel );
}

```



```

if (pi->birth)
    putstring(BIRTH ,convdat(pi->birth,1));
putstring(CODE ,pi->code);
if (pi->ndmonth)
{
    sprintf(buffer,"% d/  putstring(NAMEDAY,buffer );
}
putstring(OFFICE,pi->office );
putstring(OADDR ,pi->oaddr );
putstring(OTEL1 ,pi->otel1 );
putstring(OTEL2 ,pi->otel2 );
for (i = 0; i < pi->debtn; ++i)
    putstring(DEPTS,pi->debts[i]);
for (i = 0; i < pi->claimn; ++i)
    putstring(CLAIMS,pi->claims[i]);
putstring(NOTE ,pi->note );
}
static putstring(kw,str)
/* Egy string kiírása a megfelelő kulcsszóval */
int kw; /* a kulcsszó indexe */
char *str; /* a kiírandó string */
{
    if (str == NULL|| *str == '\0')
        return; /* nincs értéke, nem írjuk ki */
    fprintf(fp,"% s% s\n",KW[kw],str);
}
static makebakfile(filename)
/* mentő file képzése */
char *filename; /* a file neve */
{
    char bakfname[80]; /* a mentő file neve */
    if (access(filename,0)) /* a file még nem létezik? */
        {

```

```
    errno = 0;
    return;
}

strcpy(bakfname,filename);
forcefext(bakfname,"FIQ"); /* mentő file típusa: ".FIQ" */
if (access(bakfname,0) == 0) /* a mentő file már létezik? */
{
    if (unlink(bakfname)) /* eltöröljük */
    {
        errno = 0;
        sherror("A mentő file nem törölhető!",
            bakfname,NULL);
        return;
    }
}

if (rename(filename,bakfname))
{
    errno = 0;
    sherror("A(z)",
        filename,
        "nem nevezhető át",
        bakfname,
        "névre!",
        NULL);
}

}

static char *getline(pkind)
/* Egy sor beolvasása */
int      *pkind; /* a sor típusa */
{
    register int    t; /* aktuális pozíció a bufferben */
    char          **pkw; /* a kulcsszó tábla változója */
    register int    i; /* ciklusváltozó */

```



```
NewLine: /* üres vagy magyarázó sor volt */
t = 0; /* kezdő pozíció */
Again: /* lecsurgó ciklus */
if (t >= MAXBUF-1)
{
    sherror("Túl hosszú sor!",buffer,NULL);
    endism(4);
}
if (feof(fp)) /* file vége? */
{
    IsEof:
    buffer[t] = '\0'; /* sor lezárása */
    if (t == 0) /* üres sor volt? */
    {
        *pkind = ENDLINES; /* nincs több sor */
        return(buffer);
    }
    goto EndLine; /* a sor elemzése */
}
switch ((buffer[t] = getc(fp)))
{
    case EOF:
    case '\32':
        goto IsEof; /* file vége */
    case '\t':
        buffer[t] = ' '; /* a tabulátorból szóköz lesz */
        break;
    case '\n':
        if (t <= 0 || buffer[0] == '*')
            goto NewLine; /* üres vagy magyarázó sor */
        --t;
        while (buffer[t] == ' ')
            --t; /* sor végi szóközök levágása */
}
```

```
    ++t;
    if (t <= 0)
        goto NewLine; /* csak szóközöket tartalmaz */
    buffer[t] = '\0';
    goto EndLine; /* a sor elemzése */
case '\r': /* elhagyjuk */
    goto Again;
}
++t; /* következő pozíció */
goto Again;
EndLine: /* a sor elemzése */
for (t = 0; buffer[t] && buffer[t] != '='; ++t)
    ; /* kulcsszó végéig... */
if (buffer[t] == '=') /* van kulcsszó? */
{
    ++t;
    for (pkw = KW, i = 0; *pkw; ++pkw, ++i)
    {
        if (memcmp(buffer,*pkw,t) == 0)
        {
            *pkind = i;
            return(buffer+t);
        }
    }
}
sherror("Hibás sor",buffer,NULL);
/* nincs vagy rossz kulcsszó */
endism(4);
}
/* end */
```



```

/* FUNC.C
**
** A mezőkhöz rendelt beolvasó, kiíró, ellenőrző
** stb. függvények
**/

#include "ISM.EXT"

int convstring(fieldnum,addr)
/* Konvertálja a TEXT mezőben az ékezetes karaktereket */
int fieldnum; /* a mező sorszáma */
char *addr; /* mutató a belső bufferben */
{
char buffer[80]; /* konvertáláshoz */
int length;
length = Field_Length(fieldnum); /* a mező hossza */
txttostr(addr,length,buffer); /* konvertálás C stringgé */
if (buffer[0] == '\0') /* üres string? */
{
if (fieldnum == F_name) /* ha ez a név, akkor nem lehet */
return(F_UPD_AGAIN); /* üres, üsse be újra! */
Field_Def(fieldnum,NO); /* nincs érték */
return(F_UPD_OK); /* jelzés: OK */
}
tomagyar(buffer); /* az ékezetek konvertálása */
strtotxt(buffer,addr,length); /* string konvertálása
                                TEXT-té */
Field_Def(fieldnum,YES); /* a mező értéket kapott */
return(F_UPD_OK); /* az érték jó */
}

int kind_input(fieldnum,addr,length,isuns,clr,pressed)
/* A rekord szűrő-értékének (kind) beolvasó függvénye */
int fieldnum; /* a mező sorszáma */
char *addr; /* mutató a belső bufferben */
int length; /* hossz a képernyőn */

```

```

int  isuns; /* előjel nélküli? */
int  clr; /* editálási szín */
int  pressed; /* a leütött billentyű kódja */
{
int  new; /* az új érték */
if ( (new = selstr(FILTER,"Szűrő",filter)) < 0)
    return(0); /* ESC-pel visszautasítva */
if (actpi->kind == new) /* nem változott */
    return(ENTER); /* jelzés: OK */
actpi->kind = *addr = new; /* az új érték beállítása */
if (!inselected(actism)) /* az aktuális szűrőn kívüli? */
{
    if (!nextrecord())
        prevrecord();
    Form_Break(ENTER); /* process(): NewRecord */
    return(ENTER); /* jelzés: OK */
}
set_protect(); /* a mezők védettsége */
Form_Fields(); /* újra kiíratjuk a színek miatt*/
return(ENTER); /* jelzés: OK */
}
set_protect()
/* Beállítja a személyes/hivatalos mezők védettségét
** a szűrő-értéktől függően
*/
{
register int  i; /* ciklusváltozó */
for (i = F_addr; i <= F_otel2; ++i)
    Field_Protect(i,NO); /* először inputtá tesszük */
switch (actpi->kind)
{
case MIND:
    break;

```



```

case SZEMELYES:
    for (i = F_office; i <= F_otel2; ++i)
        Field_Protect(i,YES);
    break;
case KOZULET:
    for (i = F_addr; i < F_office; ++i)
        Field_Protect(i,YES);
    break;
}
}

int birth_update(fieldnum,addr)
/* A születési dátum ellenőrzése a személyi szám alapján */
int fieldnum; /* a mező sorszáma */
long *addr; /* mutató a belső bufferben */
{
    char buf[9]; /* a sz.sz-ból képzett dátumhoz */
    long dat; /* a sz.sz-ból képzett dátum-érték */
    if (*addr == 01) /* szül.dátum törölve? */
    {
        Field_Def(fieldnum,0); /* a születési dátum nincs */
        return(F_UPD_OK); /* jelzés: OK */
    }
    if (Field_Get(F_code,buffer) < 0) /* van személyi szám? */
    {
        Field_Def(fieldnum,1); /* a születési dátum jó */
        return(F_UPD_OK); /* jelzés: OK */
    }
    strcpy(buf," / / "); /* dátum forma alapja */
    buf[0] = buffer[2]; /* az év áttöltése */
    buf[1] = buffer[3]; /* - " - */
    buf[3] = buffer[4]; /* a hónap áttöltése */
    buf[4] = buffer[5]; /* - " - */
    buf[6] = buffer[6]; /* a nap áttöltése */

```

```
buf[7] = buffer[7]; /* - " - */
dat = atodat(buf,1); /* konvertálás dátummá */
if (dat != *addr)
{
    sherror("A személyi szám más születési",
           " dátumot tartalmaz!",NULL);
    return(F_UPD_BAD); /* jelzés: rossz */
}
Field_Def(fieldnum,1); /* az érték jó */
return(F_UPD_OK); /* jelzés: OK */
}

int code_update(fieldnum,addr)
/* Személyi szám ellenőrzése és a születési
dátummal való összevetése */
int fieldnum; /* a mező sorszáma */
char *addr; /* mutató a belső bufferben */
{
    register int i; /* ciklusváltozó */
    register char *s; /* a számjegyek összeadásához */
    int sum = 0; /* a számjegyek összege */
    char ic[12]; /* a tömörített sz.sz. */
    char buf[9]; /* a sz.sz-ból képzett dátum */
    long dat; /* a konvertált dátum-érték */
    long b; /* születési dátum-érték */
    /* Kivesszük a kötött karaktereket */
    Compress_Fmt("#-#####-####",addr,ic,Field_Length(fieldnum));
    /* ha mind '0', akkor ez a személyi szám törlése */
    i = NO;
    for (s = ic; s < &ic[11]; ++s)
    {
        if (*s != '0')
        {
            i = YES;
        }
    }
}
```



```

        break;
    }
}
if (i == NO)
{
    Field_Def(fieldnum,0); /* az érték megszűnt */
    return(F_UPD_OK); /* jelzés: OK */
}
s = ic;
if (*s < '1' || *s > '4')
    goto badcode; /* rossz az első számjegy */
for (i = 1; i < 11; i++)
    sum += (*s++-'0') * i; /* számjegyek összeadása */
sum %= 11; /* checksum képzés */
if (sum != *s-'0') /* az ellenőrző számjegy */
    goto badcode; /* a személyi szám hibás */
strcpy(buf," / / "); /* dátum forma alapja */
s = ic;
buf[0] = *++s; /* az év áttöltése */
buf[1] = *++s; /* - " - */
buf[3] = *++s; /* a hónap áttöltése */
buf[4] = *++s; /* - " - */
buf[6] = *++s; /* a nap áttöltése */
buf[7] = *++s; /* - " - */
dat = atodat(buf,1); /* konvertálás dátummá */
if (Field_Get(F_birth,(char *)&b) < 0)
    /* nem volt születési dátum? */
{
    Field_Set(F_birth,(char *) &dat); /* beállítjuk */
    Field_Put(F_birth); /* kiíratjuk */
    Form_Changed(F_birth,-1); /* saját változtatás! */
}
else

```

```

{
if (dat != b) /* sz.sz <-> szül.dátum */
{
sherror("A születési dátum hibás!",NULL);
return(F_UPD_BAD); /* jelzés: rossz */
}
}

Field_Def(fieldnum,YES); /* az érték jó */
return(F_UPD_OK); /* jelzés: OK */

badcode:
sherror("Hibás személyi szám!",NULL);
return(F_UPD_BAD); /* jelzés: rossz */
}

int ndmonth_update(fieldnum,addr)
/* A névnap hónapjának ellenőrzése */
int fieldnum; /* a mező sorszáma */
char *addr; /* mutató a belső bufferben */
{
if (*addr == 0) /* törölve? */
{
Field_Def(fieldnum,0); /* az érték törölve */
Field_Def(F_ndday,0); /* töröljük a napot is */
Field_Put(F_ndday);
return(F_UPD_OK); /* jelzés: OK */
}

if (*addr < 1 || *addr > 12) /* január - december */
return(F_UPD_BAD); /* jelzés: rossz */

Field_Def(fieldnum,1); /* az érték jó */
return(F_UPD_OK); /* jelzés: OK */
}

int ndday_update(fieldnum,addr)
/* A névnap napjának ellenőrzése */
int fieldnum; /* a mező sorszáma */

```



```

char *addr; /* mutató a belső bufferben */
{
if (*addr == 0) /* törölve? */
{
Field_Def(fieldnum,0); /* az érték törölve */
Field_Def(F_ndmonth,0); /* töröljük a hónapot is */
Field_Put(F_ndmonth);
return(F_UPD_OK); /* jelzés: OK */
}
if (*addr < 1 || *addr > 31) /* a rövid hónapokra
                                nem figyel */
return(F_UPD_BAD); /* jelzés: rossz */
Field_Def(fieldnum,1); /* az érték jó */
return(F_UPD_OK); /* jelzés: OK */
}
int debts_load(actitem,function)
/* A Tartozások (debts) al-form betöltő függvénye */
int actitem; /* az alrekord sorszáma */
int function; /* az aktuális funkció */
{
if (function) /* nem tényleges betöltés */
return(YES);
strtotxt(actpi->debts[actitem],buffer,Field_Length(0));
Field_Set(0,buffer); /* a mezőnek van értéke */
}
int debts_store(actitem,function)
/* A Tartozások (debts) al-form kimentő függvénye */
int actitem; /* az alrekord sorszáma */
int function; /* az aktuális funkció */
{
char str[80]; /* a megfelelő C string */
char **pp; /* mutató a dinamikus tömbben */
char *s; /* az AINSERT-hez */

```

```
if (function == -1) /* az alrekord törlése? */
{
    free(actpi->debts[actitem]);
    ADELETE(char**, actpi->debts, actpi->debtn, actitem);
    return(YES); /* a törlés megtörtént */
}

Field_Get(0, buffer); /* az alrekord értéke */
txttostr(buffer, Field_Length(0), str);

/* konvertálás C stringgé */

if (function == 1) /* beszúrás? */
{
    s = newname(buffer);
    AINSERT(char**, actpi->debts, actpi->debtn, actitem, &s);
}
else
{
    /* módosítás */
    pp = (actpi->debts)+actitem;
    if (strcmp(*pp, buffer)) /* változott */
        *pp = xchgstr(buffer, *pp);
}

return(YES); /* return: OK */
}

int claims_load(actitem, function)
/* A Követelések (claims) al-form betöltő függvénye */
int actitem; /* az alrekord sorszáma */
int function; /* az aktuális funkció */
{
    if (function) /* nem tényleges betöltés */
        return(YES);

    strtotxt(actpi->claims[actitem], buffer, Field_Length(0));
    Field_Set(0, buffer); /* a mezőnek van értéke */
}

int claims_store(actitem, function)
```



```

/* A Követelések (claims) al-form kimentő függvénye */
int  actitem; /* az alrekord sorszáma */
int  function; /* az aktuális funkció */
{
char  str[80]; /* a megfelelő C string */
char  **pp; /* mutató a dinamikus tömbben */
char  *s; /* az AINSERT-hez */
if (function == -1) /* az alrekord törlése? */
{
    free(actpi->claims[actitem]);
    ADELETE(char**,actpi->claims,actpi->claimn,actitem);
    return(YES); /* a törlés megtörtént */
}
Field_Get(0,buffer); /* az alrekord értéke */
txttostr(buffer,Field_Length(0),str);
/* konvertálás C stringgé */
if (function == 1) /* beszúrás? */
{
    s = newname(buffer);
    AINSERT(char**,actpi->claims,actpi->claimn,actitem,&s);
}
else
{
    /* módosítás */
    pp = (actpi->claims)+actitem;
    if (strcmp(*pp,buffer)) /* változott */
        *pp = xchgstr(buffer,*pp);
}
return(YES); /* return: OK */
}
/* end */

```

```

/* ASSKEYS.C
**
** Az elrendelt függvények
*/
#include <alloc.h>
#include <editline.h>
#include "ISM.EXT"

static char HELP[] = "\
' ' : Név szerinti keresés\r\n\
Alt-E: Évfordulók kiírása a képernyőre\r\n\
Alt-K: Keresések (Tartozik/Követel/Megjegyzés)\r\n\
Alt-L: telefonszámok Listázása\r\n\
Alt-M: Mentés\r\n\
Alt-N: Névnep beállítása táblázat segítségével\r\n\
Alt-S: Szűrő beállítása\r\n\
Alt-T: az aktuális Törlése\r\n\
Alt-U: új cím felvitele \r\n\
^PgUp: ugrás az első rekordra\r\n\
^PgDn: ugrás az utolsó rekordra\r\n\
^F : Szabad memória\r\n\
";

int Help(pressed,fieldnum,actrec)
/* 'F1': help */
int pressed; /* a leütött billentyű kódja */
int fieldnum; /* az aktuális mező sorszáma */
int actrec; /* az aktuális alrekord sorszáma*/
{
    printhelp(HELP,glcolors[CHHELP]);
    return(0); /* pressed key || 0 for again || -1 for screen */
}

int Search(pressed,fieldnum,actrec)
/* ' ': Név szerinti keresés */
int pressed; /* a leütött billentyű kódja */

```



```

int    fieldnum; /* az aktuális mező sorszáma */
int    actrec;   /* az aktuális alrekord sorszáma*/
{
    int        findcompare();
    register char *p; /* a név beolvasásához */
    int        act; /* munkaváltozó */
    if ( (p = scanstr("Adja meg a nevet: ",NULL,40)) == EL_ESC||
                                                p == EL_NUL)
        return(0);
    tomagyar(p); /* ékezetek konvertálása */
    if ((act = findplace(p)) == actism)
        return(0); /* az aktuálist kerestük */
    if (!inselected(act))
    {
        shmsg("A keresett név nincs a szűrőn belül!",NULL);
        return(0);
    }
    actism = act;
    Form_Break(ENTER); /* process(): NewRecord */
    return(0);
}

int Insert(pressed,fieldnum,actrec)
/* Alt-U: új rekord felvitele */
int    pressed; /* a leütött billentyű kódja */
int    fieldnum; /* az aktuális mező sorszáma */
int    actrec;   /* az aktuális alrekord sorszáma*/
{
    char *p; /* a név beolvasásához */
    p = scanstr("Adja meg a nevet: ",NULL,60);
    if (p == EL_ESC || p == EL_NUL)
    {
        if (ismn == 0) /* ha még nem volt rekord, */
            Form_Break(ESC); /* kiszállás */
    }

```

```
return(0);
}

tomagyar(p); /* ékezetek konvertálása */
strcpy(insbuffer,p); /* elmentjük a globális v.-ban */
Form_Break(pressed); /* Form_Exec() break */
return(0);
}

int Delete(pressed,fieldnum,actrec)
/* Alt-T: az aktuális rekord törlése */
int pressed; /* a leütött billentyű kódja */
int fieldnum; /* az aktuális mező sorszáma */
int actrec; /* az aktuális alrekord sorszáma */
{
if (shgetans("Biztos, hogy el akarja törölni?",NULL) == NO)
return(0);
Form_Break(pressed); /* Form_Exec() break */
return(0);
}

int Break(pressed,fieldnum,actrec)
/* Megszakítja a Form_Exec()-et
a leütött billentyű kódjával */
int pressed; /* a leütött billentyű kódja */
int fieldnum; /* az aktuális mező sorszáma */
int actrec; /* az aktuális alrekord sorszáma */
{
Form_Break(pressed);
return(0);
}

int Filter(pressed,fieldnum,actrec)
/* Alt-S: szűrő beállítása */
int pressed; /* a leütött billentyű kódja */
int fieldnum; /* az aktuális mező sorszáma */
int actrec; /* az aktuális alrekord sorszáma */
```



```

{
int  new; /* az új szűrő-érték */
if ( (new = selstr(FILTER,"Szűrő",filter)) < 0)
    return(0); /* ESC-pel visszautasítva */
filter = new;
if (inselected(actism)) /* Az aktuális a szűrőn belül? */
    return(0);
actism = -1; /* Az első szűrőn belüli keresése */
nextrecord();
Form_Break(ENTER); /* process(): NewRecord */
return(0);
}

typedef int      (*FVP)();
static  char  *QUERY[] = {
    "Bármely (Tartozik-Követel-Megjegyzés)",
    "Elszámolandó (Tartozik-Követel)",
    "Tartozik",
    "Követel",
    "Megjegyzés",
    NULL};

static  int  qany(),qbal(),qdebt(),qclaim(),qnote();
static  FVP  QFVP[] = {qany,qbal,qdebt,qclaim,qnote};
static  int  oldquery = 0;

int Query(pressed,fieldnum,actrec)
/* Alt-K: keresések */
int  pressed; /* a leütött billentyű kódja */
int  fieldnum; /* az aktuális mező sorszáma */
int  actrec; /* az aktuális alrekord sorszáma */
{
int  q; /* keresési kritérium */
FVP  fp; /* a hívandó függvény táblából */
int  act; /* ciklusváltozó */
if ( (q = selstr(QUERY,"Keresések",oldquery)) < 0)

```

```

    return(0); /* ESC-pel visszautasítva */
fp = QFVP[q];
for (act = actism+1; act < ismn; ++act)
{
    if (!inselected(act)) /* nincs a szűrőben */
        continue;
    if ((*fp)(act)) /* a megfelelő függvény */
    {
        oldquery = q; /* elmentés a következőhöz */
        actism = act; /* ráállítás */
        Form_Break(ENTER); /* process(): NewRecord */
        return(0);
    }
}
return(0);
}

static int qany(act)
/* Megvizsgálja, hogy az aktuálisban van-e
   Tartozik-Követel-Megjegyzés */
{
    PISM pi;
    pi = isms+act;
    return(pi->debts != NULL ||
        pi->claims != NULL ||
        pi->note != NULL);
}

static int qbal(act)
/* Megvizsgálja, hogy az aktuálisban van-e Tartozik-Követel */
{
    PISM pi;
    pi = isms+act;
    return(pi->debts != NULL || pi->claims != NULL);
}

```



```
static int qdebt(act)
/* Megvizsgálja, hogy az aktuálisban van-e Tartozik */
{
PISM pi;
pi = isms+act;
return(pi->debts != NULL);
}

static int qclaim(act)
/* Megvizsgálja, hogy az aktuálisban van-e Követel */
{
PISM pi;
pi = isms+act;
return(pi->claims != NULL);
}

static int qnote(act)
/* Megvizsgálja, hogy az aktuálisban van-e Megjegyzés */
{
PISM pi;
pi = isms+act;
return(pi->note != NULL);
}

int FirstRec(pressed,fieldnum,actrec)
/* Ctrl-PgUp: ugrás az első rekordra */
int pressed; /* a leütött billentyű kódja */
int fieldnum; /* az aktuális mező sorszáma */
int actrec; /* az aktuális alrekord sorszáma */
{
if (actism == 0)
return(0);
actism = -1;
nextrecord();
Form_Break(ENTER); /* NewRecord */
return(0);
```

```

}

int LastRec(pressed,fieldnum,actrec)
/* Ctrl-PgDn: ugrás az utolsó rekordra */
/* Positions on the last record */
int   pressed; /* a leütött billentyű kódja */
int   fieldnum; /* az aktuális mező sorszáma */
int   actrec; /* az aktuális alrekord sorszáma */
{
if (actism == ismn-1)
    return(0);
actism = ismn;
prevrecord();
Form_Break(ENTER); /* >j rekord */
return(0);
}

int FreeMem(pressed,fieldnum,actrec)
/* A szabad memóri kiírása */
int   pressed; /* a leütött billentyű kódja */
int   fieldnum; /* az aktuális mező sorszáma */
int   actrec; /* az aktuális alrekord sorszáma */
{
#ifdef __TINY__ || defined(__SMALL__) ||
defined(__MEDIUM__)
sprintf(buffer,"Szabad memória: % u",coreleft());
#else
sprintf(buffer,"Szabad memória: % lu",coreleft());
#endif
shmsg(buffer,NULL);
return(0);
}

/* end */

```



```

/* MISC.C
**
** Kiegészítő rutinok
*/
#include "ISM.EXT"
nextrecord()
/* Pozícionálás a következő, a szűrőnek megfelelő rekordra */
{
register PISM pi; /* az aktuális rekord mutatója */
register int act; /* az aktuális rekord sorszáma */
if (actism >= ismn-1) /* az utolsón állunk? */
    return(NO); /* nincs következő */
act = actism+1; /* a következőtől keresünk */
pi = isms+act; /* a rekord mutatója */
while (!inselected(act)) /* amíg nincs a szűrőben... */
{
    if (++act >= ismn) /* az utolsó után? */
        return(NO); /* nincs következő */
    ++pi;
}
actism = act; /* áttesszük a globálisba */
return(YES);
}
prevrecord()
/* Pozícionálás az előző, a szűrőnek megfelelő rekordra */
{
register PISM pi; /* az aktuális rekord mutatója */
register int act; /* az aktuális rekord sorszáma */
if (actism <= 0) /* az elsőn állunk? */
    return(NO); /* nincs előző rekord */
act = actism-1; /* az előzőtől keresünk */
pi = isms+act; /* a rekord mutatója */
while (!inselected(act)) /* amíg nincs a szűrőben... */

```

```
{
    if (--act < 0) /* az első előtt? */
        return(NO); /* nincs előző rekord */
    --pi;
}

actism = act; /* áttesszük a globálisba */
return(YES);
}

inselected(rec)
/* Megvizsgálja, hogy a megadott rekord megfelel-e a
   szűrőnek */

int rec; /* a vizsgálandó sorszáma */
{
    switch (filter)
    {
        case MIND: /* bármelyik jó */
            return(YES);
        case SZEMELYES: /* csak a nem közület jó */
            return( (isms+rec)->kind != KOZULET);
    }
    return( (isms+rec)->kind == KOZULET); /* csak a közület jó */
}

int findplace(name)
/* Megkeresi a nevet a rekordok között */
char *name; /* a keresendő név */
{
    register PISM pi; /* rekord mutató */
    register int i; /* ciklusváltozó */
    for (pi = isms, i = 0; i < ismn; ++pi, ++i)
    {
        if (sacompere(name, pi->name) <= 0)
            break;
    }
}
```



```

/* ANNO.C
**
** Kíírja a képernyőre a közeli szül-, és névnapokat
*/
#include <window.h> /* CEX header */
#include <frame.h> /* CEX header */
#include <datday.h> /* CEX2 header */
#include "ISM.EXT"
#define LIN      3 /* kezdő sor */
extern char      numbuf[]; /* a convnum() buffere */
int Anno()
/* Alt-E: a szül-, és névnapok listázása */
{
WINDTYPE      annoframe();
WINDTYPE      sw; /* a lementett ablak */
register PISM   pi; /* aktuális rekord */
register int    i; /* ciklusváltozó */
union UDATE    dat; /* a dátum számításokhoz */
int            actyear; /* aktuális év */
long           actdays; /* aktuális napok száma */
int            birnum = 0; /* születésnapok száma */
int            namnum = 0; /* névnapok száma */
long           diff; /* eltérés az actdays-től */
sysdate(&dat.s); /* az aktuális dátum beszédese */
actdays = dattoday(dat.l); /* az aktuális napok
                             számítása */
actyear = dat.s.year; /* az aktuális évet külön */
for (pi = isms, i = 0; i < ismn; ++pi, ++i)
{
    if (pi->kind == KOZULET)
        continue; /* csak ha van személyes info */
    if (pi->birth) /* van születésnap? */
    {

```



```

dat.l = pi->birth;
dat.s.year=actyear; /* az év átállítása a mostanira */
diff = dattoday(dat.l) - actdays;
if (diff > -201 && diff < 201) /* 20 napon belül van? */
{
    if (birnum == 0 && namnum == 0)
        sw = annoframe(); /* ez az első */
    cupo(birnum+LIN,2);
    aputtysn(pi->name,GRAY,26);
    cupo(birnum+LIN,28);
    aputtys("19",GRAY);
    aputtys(convdat(pi->birth,1),GRAY);
    birnum++;
}
}

if (pi->ndmonth) /* van névnap? */
{
    dat.s.year = actyear;
    dat.s.month = pi->ndmonth;
    dat.s.day = pi->nnday;
    diff = dattoday(dat.l) - actdays;
    if (diff > -201 && diff < 201) /* 20 napon
                                   belül van? */
    {
        if (birnum == 0 && namnum == 0)
            sw = annoframe(); /* ez az első */
        cupo(namnum+LIN,41);
        aputtysn(pi->name,GRAY,26);
        cupo(namnum+LIN,69);
        aputtys(months[dat.s.month],GRAY);
        aputtys(" ",GRAY);
        convnum(dat.s.day);
        aputtys(numbuf+4,GRAY);
    }
}

```

```
        namnum++;
    }
}

if (birnum == 0 && namnum == 0) /* nem volt évforduló */
{
    shmsg("Nincs évforduló.",NULL);
    return(0);
}

cupo(24,0);
aputtys("Üssön le egy billentyűt a folytatáshoz...",GRAY);
kbget(); /* várakozás */
windclose(sw); /* a képernyő vissza */
return(0);
}

static WINDTYPE annoframe()
/* A képernyő lementése után kiírja az évfordulók keretét */
{
    WINDTYPE sw;
    if ( (sw = windsave(0,0,24,79)) == (WINDTYPE)0)
    {
        erase();
        puttys("Nincs elég memória!");
        exit(4);
    }

    erase();
    qframe(0,0,23,78,0,GRAY);
    colsimple(0,39,24,GRAY);
    linsimple(LIN-1,0,79,GRAY);
    cupo(1,12);
    aputtys("Születésnapok",GRAY);
    cupo(1,53);
    aputtys("Névnapok",GRAY);
```



```
return(sw);  
}  
/* end */
```

```

/* NAMEDAY.C
**
** Névnep kiválasztása az ISM.ND file-ban lévő táblázattal,
** a file első sorában a táblázat elemeinek száma van,
** a file további sorainak formája: keresztnév,hónap,nap.
*/

#include <dselect.h>
#include "ISM.EXT"
typedef struct ND
{
    char *n; /* keresztnév */
    char m; /* hónap */
    char d; /* nap */
} *PND, SND;

static int ndn = 0; /* a táblázat elemeinek száma */
static PND nds = NULL; /* a táblázat */

int NameDay(pressed,fieldnum,actrec)
/* Alt-N: Névnep kiválasztása táblázatból */
int pressed; /* a leütött billentyű kódja */
int fieldnum; /* az aktuális mező sorszáma */
int actrec; /* az aktuális alrekord sorszáma */
{
    int toscr(); /* dselect()-nek kiíró fv. */
    PND pnd; /* kereséshez mutató */
    int i; /* ciklusváltozó */
    register char *s; /* munkaváltozó */
    register char *p; /* munkaváltozó */
    char name[80]; /* a név C string-ként */
    char old; /* korábbi hónap, ill. nap */
    if (Field_Get(F_kind,&old) < 0 || old == KOZULET)
        return(0); /* közületnek nincs névnepja */
    if (Field_Get(F_name,buffer) < 0)
        return(0); /* a név nincs még definiálva */

```



```

txttostr(buffer,Field_Length(F_name),name);
if ( (s = strchr(name,' ')) == NULL)
    return(0); /* nincs keresztnév */
++s; /* a keresztnév kezdete */
for (p = s; *p && *p != ' ' && *p != ','; ++p)
    ;
*p = '\0'; /* csak az első keresztnévre */
if (nds == NULL) /* ha a táblázat nincs beolvasva */
    if (readnds()) /* beolvassuk */
        return(0); /* rossz olvasás */
/* Megkeressük a táblázatban a név első előfordulását */
for (pnd = nds, i = 0; i < ndn; ++pnd, ++i)
{
    if ( (old = sacompare(pnd->n,s)) == 0)
        break;
    if (old > 0)
        break;
}
if (old != 0)
{
    sherror("A keresztnév nem szerepel a táblázatban",s,NULL);
    return(0);
}
i = dselect(ndn,
    30,
    i,
    toscr,
    "Names",
    glcolors[CHELP],
    glcolors[CMMSG],
    (int (*)( ))0,
    NULL);
if (i < 0)

```

```

    return(0); /* ESC-pel visszautasítva */
pnd = nds+i; /* a táblázatbeli elem */
if (Field_Get(F_ndmonth,&old) < 0 || old != pnd->m)
{
    /* nem ugyanaz a hónap */
    Field_Set(F_ndmonth,&pnd->m); /* beállítás */
    Field_Put(F_ndmonth); /* kiíratás */
    Form_Changed(F_ndmonth,-1); /* saját változtatás! */
}
if (Field_Get(F_ndday,&old) < 0 || old != pnd->d)
{
    /* nem ugyanaz a nap */
    Field_Set(F_ndday ,&pnd->d);
    Field_Put(F_ndday);
    Form_Changed(F_ndday,-1);
}
return(0);
}

static char NDFILE[] = "ISM.ND"; /* a táblázatot
                                   tartalmazó file */

static int readnds()
/* Beolvassa az ISM.ND file-ból a névnapok táblázatát */
{
    FILE *fp;
    int i;
    register PND pnd;
    register char *s;
    if ( (fp = fopen(NDFILE,"r")) == NULL)
    {
        sherror(NDFILE,NULL);
        return(-1);
    }
    fgets(buffer,80,fp); /* a táblázat elemeinek a száma */
    if ( (ndn = atoi(buffer)) <= 0)
    {

```



```

sherror("Üres file",NDFILE,NULL);
return(-1);
}

pnd = nds = (PND) myalloc(ndn * sizeof(SND));
for (i = 0; i < ndn; ++i)
{
    if (fgets(buffer,80,fp) == NULL)
    {
        sherror("Hibás file",NDFILE,NULL);
        exit(4);
    }
    s = strchr(buffer,',');
    *s++ = '\0';
    pnd->n = newname(buffer);
    pnd->m = atoi(s);
    pnd->d = atoi(strchr(s,',')+1);
    ++pnd;
}

return(0); /* nincs hiba */
}

static int toscr(act,lin,col)
/* A dselect() fv. számára kiíró fv. */
{
    extern char numbuf[]; /* a convnum() buffere */
    register PND pn; /* a táblázatbeli elem */
    pn = nds+act;
    cupo(lin,col);
    aputtysn(pn->n,glcolors[CHELP],23);
    cupo(lin,col+24);
    aputtys(months[pn->m],glcolors[CHELP]);
    convnum(pn->d);
    aputtys(numbuf+3,glcolors[CHELP]);
}

/* end */

```

```
/* REPORT.C
**
** A szűrőben lévő rekordokból telefonkönyv listázása
*/
#include "ISM.EXT"
#define MAXPAGE 65 /* az egy lapra írható sorok száma */
static char lstname[] = "ISM.LST"; /* a listafile neve */
int Report()
/* Alt-L: A telefonszámok listájának generálása */
{
char          actdat[12]; /* a fejlécbe kerülő dátum */
int           listed = MAXPAGE+1; /* a lapra már
                                   kiírt sorok száma*/
int           pnun = 0; /* az aktuális lap sorszáma */
FILE          *fp; /* az output file */
register PISM pi; /* az aktuális rekord mutatója */
register char *p; /* az otthoni telefon */
char          *p1; /* a hivatali 1. telefon */
char          *p2; /* a hivatali 2. telefon */
int           saveactism; /* az aktuális rekord sorszáma */
char          *EMPTY = ""; /* többször használt
                                   üres string*/

if (shgetans("A szűrő be van állítva?", NULL) == NO)
    return(0);

if ((fp = fopen(lstname, "w")) == NULL)
{
    sherror(lstname, NULL);
    return(0);
}

saveactism = actism; /* elmentjük a visszaállításhoz */
getdat(actdat); /* az aktuális dátum */
actism = -1; /* az elsőről fogunk indulni */
while (nextrecord())
```



```

{
    statline(actism+1); /* itt tartunk a generálásban */
    if (++listed >= MAXPAGE) /* tele van az aktuális lap? */
    {
        fprintf(fp, "\f% -71sPage: % d\n\n", actdat, ++pnum);
        listed = 3;
    }
    pi = isms+actism;
    if (pi->tel == NULL && pi->otel1 == NULL &&
        pi->otel2 == NULL)
        continue; /* nincs telefonszám */
    if ((p = pi->tel) == NULL)
        p = EMPTY;
    if ((p1 = pi->otel1) == NULL)
        p1 = EMPTY;
    if ((p2 = pi->otel2) == NULL)
        p2 = EMPTY;
    if (p1 != EMPTY && p2 != EMPTY)
    {
        if (strncmp(p1, p2, 8) == 0) /* ha a fővonal ua., */
        {
            /* akkor összevonjuk */
            strcpy(buffer, p1);
            strcat(p1 = buffer, ", ");
            strcat(p1, p2+8);
            p2 = EMPTY;
        }
    }
    fprintf(fp, "% -16.16s% -32.32s% s,
        "% s\n", p, pi->name, p1, p2);
}
if (ferror(fp)) /* volt írási hiba? */
{
    sherror(1stname, NULL);
}

```

```
fclose(fp);
unlink(lstname);
}
else
{
fclose(fp);
shmsg("A generált telefonkönyv az",lstname,
      "file-ban.",NULL);
}
actism = saveactism; /* visszaállunk az eredetire */
statline(actism+1); /* a százalékos kijelzés */
return(0);
}
/* end */
```


5.5.4. Megjegyzések a példaprogramhoz

A program két paramétert értelmelmez:

- b, fekete-fehér üzemmód kérése;
- a, csak az évfordulók (születés- és névnapok) kiírását kérjük.

A program nagymértékben támaszkodik a CEX és a CEX2 nevű C könyvtárakra, ezért a program működésének megértése feltételezi e könyvtárak ismeretét.

Bizonyos funkciók (pl. Report, Query) itt rögzítetten és beprogramozva találhatók, bár ezekre is léteznek külön rendszerek, amelyekkel a funkciók kivitelezése magasabb fokon végezhető el.

A program jelenlegi formájában a Borland TurboC 2.00 környezettel reprodukálható. Más támogatott C nyelvi környezethez a **FreeMem()** függvényt kell a megfelelő nyelvi környezethez igazítani, mivel a szabad memória lekérdezése az egyes környezetekben más és más.

A futtatható program előállítás a következő lépésekben történik:

1. Az ISM.FTX form-file alapján a FORMEDIT.EXE programmal az ISM.FH (ALT-H funkció), az ISM.FCT (ALT-F funkció) és az ISM.FIC (ALT-I funkció) file-ok létrehozása.
2. A következő paranccsal a fordítás - szerkesztés elvégzése:

```
tcc -w-par -ml -Iti -Ltl -DTURBOC -eISM *.c form1.lib
cex2lh.lib cex1.lib
```

ahol:

- ti, a TurboC header-file-jainak katalógusa, amelyben benne kell lenniük a CEX és a CEX2 header-file-oknak is;
- tl, a TurboC könyvtárak katalógusa, amelyben benne kell lenniük a FORM, a CEX és a CEX2 könyvtáraknak, a memóriamodellnek megfelelően (a memóriamodell kiváltható például -ml helyett -lc opció és a könyvtárak: formc.lib, cexc.lib és cex2ch.lib).

5.6. Hibakezelés

Az M-FORM futtatórendszer a hibákat visszatérési kódokkal jelzi. Ezen kívül az egyértelműen programozási hibákra a futtatórendszer megszakítja a program futását (pl. ha egy mező kezelő függvény olyan mezősorszámot kap paraméterként, amely nem létező mezőt azonosít).

6. fejezet

A FormDoc

A programozói és felhasználói dokumentációk fontos része a felhasználói interfész leírása. Az ezt megvalósító form-ok dokumentálását végzi a FORMDOC.EXE program.

A FORMDOC egy form-file-ban lévő összes form-ra generálja a dokumentációt a következő elemekkel:

- a form általános adatai (név, képernyőn elfoglalt hely, leírás),
- a form képe (a form minden oldala),
- az egyes mezők jellemzői,
- az elrendelt billentyűk és hozzátartozó C függvények.

A FORMDOC által generált dokumentációnak a következő variációs lehetőségei vannak:

- angol vagy magyar nyelvű (magyarázó szövegek),
- IBM képernyő forma vagy DOG forrás forma,
- önálló vagy beszúrt DOG forrás,
- DOG fejezetmélység.

A program hívása:

```
>FORMDOC form-file [-o output] [-d] [-i] [-a] [-l level] [-m]
```

ahol:

<i>form-file</i>	a form-file neve;
<i>output</i>	az output neve (default: form-file.FDC);
<i>-d</i>	DOG forrás formátum (default: IBM képernyő forma);
<i>-i</i>	beszúrt DOG forrás (default: önálló DOG forrás);
<i>-a</i>	a dokumentáció a létező outputhoz hozzáfűzött legyen (default: felülírás);
<i>level</i>	DOG fejezetmélység (1-5, default: 1 - \$CPT.);
<i>-m</i>	magyar nyelvű magyarázatok (default: angol nyelvű).

7. fejezet

Operátori kezelés

Az adatbevitel a **Form_Exec()** függvény programból történő hívásával kezdődik.

A futtatórendszer számára az egyes billentyűk leütése a következőket jelenti (az elrendelt billentyű a neki megfelelő elrendelt függvény meghívását jelenti):

ESC	az aktuális rekord bevitelének befejezése (a Form_Exec() működésének megszakítása ESC kóddal),
Left	pozícionálás a sorrendben előző input mezőre,
Up	pozícionálás a sorrendben előző input mezőre,
Right	pozícionálás a sorrendben következő input mezőre,
Down	pozícionálás a sorrendben következő input mezőre,
Home	pozícionálás az első input mezőre,
End	pozícionálás az utolsó input mezőre,
PgUp	az aktuális rekord bevitelének befejezése és az előző rekord kérése (a Form_Exec() működésének megszakítása SCAN(PGUP) kóddal),
PgDn	az aktuális rekord bevitelének befejezése és a következő rekord kérése (a Form_Exec() működésének megszakítása SCAN (PGDN) kóddal),
ENTER	az aktuális mező értékadásának kezdeményezése,
egyéb	az aktuális mező értékadásának megkezdése a leütött billentyűvel.

Ha a felsorolt billentyűk valamelyike elrendelt billentyű, akkor a fenti funkció elvégzése helyett a megfelelő elrendelt függvény hívódik meg.

Ha az operátori kezelés során egy SUBFORM mezőn belül vagyunk, akkor néhány billentyű funkciója a következőképpen módosul:

Up	pozícionálás az előző alrekord megfelelő input mezőjére,
Down	pozícionálás a következő alrekord megfelelő input mezőjére,
Home	ha az al-form nem első input mezőjén állunk: pozícionálás az al-form első input mezőjére,
End	ha az al-form nem utolsó input mezőjén állunk: pozícionálás az al-form utolsó input mezőjére,
PgUp	ha az aktuális alrekord nem az első alrekord: lapozás felfelé az alrekordokon,
PgDn	ha az aktuális alrekord nem az utolsó alrekord: lapozás lefelé az alrekordokon,
INS	új alrekord felvitele (beszúrása) az aktuális alrekord elé,
A	új alrekord felvitele az utolsó alrekord után,
DEL	az aktuális alrekord törlése,
ESC	ha egy új alrekord felvitelkor az alrekordot még nem hagytuk el, akkor a felvitel visszavonása.

Ha az adatbevitel során egy olyan SUBFORM mezőre lépünk, amely az adott pillanatban nem tartalmaz definiált alrekordokat, akkor bármely, az al-form-hoz tartozó mező értékadása egyben egy alrekord létrehozását is jelenti.

8. fejezet

A disztributív lemez tartalma

Az M-FORM disztributív lemezekben a következő file-ok találhatók:

1. lemez:

FORMEDIT.EXE	a FormEditor,
FORMEDxx.SRF	a FormEditor prototípus file-jai,
FORMDOC.EXE	a FormDocumenter,
SAMPLE.C	az egyszerű példaprogram forrása,
<ISM>	a teljes példaprogram forrásai;
ANNO.C	évfordulók kezelése,
ASSKEYS.C	az elrendelt függvények,
FUNC.C	a mezők ellenőrző stb. függvényei,
ISM.C	a főprogram,
MISC.C	kiegészítő rutinok,
NAMEDAY.C	névnap kiválasztása táblázatból,
REPORT.C	telefonkönyv generálása,
ISM.DAT	konstansok, típusok és adatdefiníciók,
ISM.EXT	konstansok, típusok és adatdefiníciók
extern változata,	
ISM.FTX	a form-file,
ISM.ND	a névnapok táblázatát tartalmazó file.

2. lemez:

FORM.H	az M-FORM header-file,
<MSC>	Microsoft C könyvtárak;
LFORM.LIB	large memóriamodell,
MFORM.LIB	medium memóriamodell,
SFORM.LIB	small memóriamodell,

<TURBOC>	TurboC könyvtárak;	
	FORML.LIB	large memóriamodell,
	FORMM.LIB	medium memóriamodell,
	FORMS.LIB	small memóriamodell.

Amennyiben Önnek az Aztec C vagy a QuickC bármely modelljének, ill. a Microsoft C vagy a TurboC medium modelljének megfelelő könyvtára van szüksége, kérjük jelleze igényét címünkön:

Magiszter Számítástechnikai Szerkesztőség

1112 Budapest, Bonc u. 3.

A kívánt könyvtárakat a szoftverkövetés keretein belül díjmentesen postafordultával küldjük.

AKADÉMIAI KIADÓ ÉS NYOMDA VÁLLALAT

Felelős kiadó: Hazai György főigazgató

Felelős szerkesztő: Juhász Lehel

A borítót Szabadi Zoltán tervezte

ISBN 963 05 5528 X

ISSN 0238-3896

A kézirat lezárva: 1989. április

Készült a TYPOTEX GT közreműködésével

