

Magiszter

Számítástechnikai szerkesztőség

CFIO MS C 4.0

alatti
FILE-KEZELŐ KÖNYVTÁR

Programvédelem nélkül!

Szoftver
IBM, PC, XT, AT
kompatibilis mikroszámítógépekre



AKADÉMIA KIADÓ



MARJAI GY.—MARJAI T.

CFIO

File-kezelő eljárások

1708 189593

MARJAI GYÖRGY—MARJAI TAMÁS

CFIO

File-kezelő eljárások

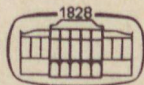
Microsoft C (4.00 változat)

nyelvhez

2.1 Változat

MS-DOS 2.XX

MS-DOS 3.XX



Akadémiai Kiadó, Budapest 1987

MTAK



660921

Szoftver-dokumentáció

Sorozatszerkesztő: Pintér Tibor

A kiadványt szakmailag ellenőrizte: Meggyesházi János

©Akadémiai Kiadó és Nyomda · Marjai György—Marjai Tamás, 1987

Minden jog fenntartva

MAGYAR
TUDOMÁNYOS AKADÉMIA
KÖNYVTÁRA

A Szoftver-dokumentáció a DOG dokumentáció generátorral készült

©Akadémiai Kiadó és Nyomda · Schill—Szegedi, 1987

M. TUD. AKADÉMIAI KÖNYVTÁRA
Könyvleltár 13062/87. SZ.

T a r t a l o m

1. Bevezetés.....	7
2. A rendszer működése, a CFIO file-ok felépítése.....	10
2.1. A CFIO file-ok felépítése.....	10.
2.2. Több szempont szerinti elérés.....	11
2.3. Az CFIO használata felhasználói programból.....	11
3. Alkalmazási lehetőségek.....	14
3.1. Kezelhető feladattípusok.....	14
3.2. Kezelhető állományszerkezetek.....	14
3.3. Más rendszerek által készített állományok gyors. kezelése	15
4. A rendszer szolgáltatásai.....	16
4.1. Beépített hibakövető rendszer.....	16
4.2. Az indexfile felépítésének vizsgálata.....	17
4.3. Sérült indexállományok helyreállítása.....	17
4.4. A sok üres rekordot tartalmazó állományok..... tömörítése	18
5. A rendszerben használható eljárások és a CFIO.H.....	19
header file	
5.1. A CFIO.H és a rendszerben használt változók.....	19
5.2. FCREAT.....	21
5.3. FIOOPEN.....	22
5.4. FIOCLOSE.....	23
5.5. SEARCH.....	24
5.6. INSERT.....	26
5.7. DELETE.....	27
5.8. NEXT.....	29
5.9. BUILD.....	31
5.10. PACK.....	33
5.11. REREAD.....	35
5.12. REWRITE.....	36
5.13. KULCSVAL.....	38
5.14. ERRORW.....	39
6. Utility programok.....	41
6.1. DUMP.....	41
6.2. BUILD.....	42
6.3. PACK.....	44
7. A CFIO rendszer DEMO programja.....	46
7.1. A DEMO.EXE program.....	46
7.2. A DEMO működése.....	47
7.3. A DEMO program forrásnyelvi listája.....	52

8. A CFIO rendszer hibajelzései.....	82
8.1. Az eljárások hibaüzenetei.....	82
8.2. A utility programok hibajelzései.....	98

1. Bevezetés

A felhasználói rendszerek készítői egyre gyakrabban térnek át az egyik legkorszerűbb programozási nyelv a 'C' használatára. Ennek igen sokszor az képezi akadályát, hogy a nyelv csak az állománykezelési alapfunkciókat tartalmazza, így további szoftver termékek beszerzése után válik csak lehetővé az állományok kulcs szerinti kezelése, illetve több szempont szerinti lekérdezése.

Ebben a témakörben megfogalmazható feladatok, illetve igények igen sokfélék lehetnek. A dokumentációban ismertetendő 'C' felhasználói könyvtár elsősorban azt az állományszerkezetet támogatja, ahol a karbantartás során az azonos kulcsú rekordok nem kerülnek be az állományba. A lekérdezés során azonban olyan lekérdezési kulcs is megadható, amely többször szerepel egy állomány rekordjaiban. Természetesen azonos kulcsú rekordok beillesztése is megengedett, de nincs definiálva, hogy a már meglévő azonos kulcsú rekordokhoz beérkezett új rekord ezek elé, mögé vagy pedig közéjük kerül. Ez azonban gyakorlatunk szerint nem jelent problémát a szükséges adatszerkezetek programozásánál.

A CFIO segítségével kezelhető állományoknak bizonyos követelményrendszert ki kell elégíteniük:

- az adatállomány csak fix hosszúságú rekordokat tartalmazhat,
- a rekordok száma nem haladhatja meg a 65024-et,
- a kulcs hossza nem haladhatja meg a 254 byte-ot,
- egy kulcs legfeljebb 20 darabból építhető össze,
- az állomány rekordjainak első karaktere nem lehet hexa 7C (függőleges vonal), mert ez az üres rekordot jelenti,
- a kulcsmező nem tartalmazhat hexa 7B karaktereknél nagyobbakat,

- az állomány nem tartalmazhat állományvége (EOF) jelet.

A rendszer működéséhez szükséges szoftver és hardver környezet:

- bármilyen IBM kompatibilis PC/XT/AT konfiguráció,
- legalább egy lemezegység,
- memóriaigény legfeljebb 170 kbyte,
- a könyvtár a Microsoft "C" 4.0 small modell lib file-jait használja a szerkesztési fázisban.

A CFIO disztributív lemezen a következő könyvtárak találhatók:

DEMO, LIB, UTILS

A könyvtárak tartalma:

- DEMO

DEMO.C	a rendszer használatát bemutató program
DEMO.EXE	forrásnyelvi és futtatható változata

- LIB

CFIO.H	a header file
CFIO.LIB	a file-kezelő rutinok run-time könyvtára

- UTILS

DUMP.EXE	az indexfile felépítését megjelenítő program
BUILD.EXE	párbeszédes indexépítő program
PACK.EXE	párbeszédes file-tömörítő program

A CFIO könyvtár a Microsoft C 4.00 SMALL memóriamodellben használható.

A CFIO könyvtár beépítése értelemeszerű: a felhasználó tárgykódú moduljaihoz hozzá kell szerkeszteni (linkelni).

2. A rendszer működése, a CFIO file-ok felépítése

2.1. A CFIO file-ok felépítése

A CFIO file-ok tulajdonképpen mindig két állományt jelentenek. Az egyik az adatokat tartalmazó file, a másik pedig a rendezett indexeket tartalmazza. A CFIO függvények változtatásokat csak az úgynevezett indexállományon végeznek. Az adatrekordokat tartalmazó állomány a feldolgozás során csak az új rekordok állományvégre függesztésével, illetve a törlendő rekordok hexa 7C karakterekkel (függőleges vonal) való felülírásával módosul.

Az indexfile sérülése esetén, vagy ha az állományt más módon, nem az indexfile segítségével módosítottuk, akkor az indexfile a build függvény használatával helyreállítható. A többi függvény helyes működéséhez nélkülözhetetlen az 'update' állapotú indexállomány.

A gyorsabb elérés érdekében az indexállomány egy kétszintű kezelési táblázatot tartalmaz. Az indexállományt legfeljebb 1022 adatrekord címének a tárolására alkalmas lapok alkotják. Egy indexállomány legfeljebb 128 lapból állhat. Technikai okok miatt azonban ezek a lapok nem lehetnek teljesen tele. Az egyidejűleg kezelhető rekordok száma 65024.

Az állomány legkisebb (vagy legnagyobb) kulcsú rekordjának címe mindig az első lap legelső címe. Utána a rekordok rendezettségi sorrendjében következnek a lapon található rekordok címei.

A többi, rekordokat tartalmazó lapról az indexállomány fejrészeben egy olyan nyilvántartás található, amely a lapokon található legkisebb (vagy legnagyobb) rekord kulcsát és a lap fizikai sorszámát tartalmazza a lapok logikai sorrendjében. Mivel a legelső adagot mindig az 1-es fizikai sorszámú lap tartalmazza, így erről nem található információ a fejrészben. A keresés az indexállományon belül kétszintű. Először a fejrészben található kulcsok segítségével megállapítjuk, hogy

a keresett kulcs melyik lapon található, majd az így kiválasztott lapot beolvassa a lapon található rendezett címek között keressük a megadott kulcsot.

2.2. Több szempont szerinti elérés.

Egy adatállományhoz több indexállomány is tartozhat. Ezek a file-kezelés szempontjából tulajdonképpen páronként más és más állományt jelentenek. Lényeges azonban, hogy ha az adatállományt az egyik indexállomány szerint karbantartottuk, de a másik szerint szeretnénk a rekordokat elérni, akkor előbb újjá kell építeni (build) ezt az indexállományt is.

2.3. Az CFIO használata felhasználói programból

A könyvtár használatához szükség van a CFIO.H header file beépítésére a felhasználói programba. A header file tartalmaz egy `'struct fileio'` deklarációjú fkl tömböt. Ha a header file-t változatlanul hagyjuk, akkor legfeljebb 20 CFIO struktúrájú állományt használhatunk. Az fkl tömböt először fel kell tölteni a CFIO állományt leíró paraméterekkel, ettől kezdve a CFIO állományt elegendő az fkl tömbben elfoglalt sorszámaival azonosítani.

Egyszerűbb azonban az fkl tömböt kezdeti értékadással feltölteni, de ehhez némileg meg kell változtatni a header file-t. Ezt egy mintafeladaton keresztül mutatjuk be.

A feladatunk legyen olyan, hogy két CFIO állományra van szükségünk.

Az egyik állomány legyen a "teszt.dat".

- Az indexfile neve "teszt.ind",
- a rekordok hossza 150 byte,
- a rendezettség kulcs szerint növekvő,

- a kulcs két részletből tevődik össze,
- az első darabja a 17. pozíción kezdődik és 6 karakter hosszú,
- a második darabja a 0.pozíción kezdődik és 4 karakter hosszú.

A másik állomány legyen a "minta.dat".

- Az indexfile neve "minta.ind",
- a rekordok hossza 90 byte,
- a rendezettség kulcs szerint csökkenő,
- a kulcs három részletből tevődik össze,
- az első darabja a 21. pozíción kezdődik és 2 karakter hosszú,
- a második darabja a 10.pozíción kezdődik és 3 karakter hosszú.
- a harmadik darabja a 8.pozíción kezdődik és 4 karakter hosszú.

A CFIO.H file-ból vegyük ki az fkl tömb deklarációját. A külső változók között, a main() előtt írjuk le a következő deklarációt. (Ld. CFIO.H struct fileio fkl deklarációját!)

```
struct fileio fkl[] =
{
    {"teszt.dat","teszt.ind",1,150,2,17,6, 0,4},
    {"minta.dat","minta.ind",0, 90,3,21,2,10,3,8,4}
};
```

Ezzel a módszerrel gondoskodhatunk róla, hogy csak a szükséges helyet foglaljuk le az fkl tömb számára.

Ettől kezdve a 0 indexű CFIO struktúra adatállománya a "teszt.dat", az 1 indexűé pedig a "minta.dat" stb.

Célszerű a

```
#define TESZT 0  
#define MINTA 1
```

utasításokkal a program elején a CFIO struktúrákhoz nevet rendelni. Így a későbbi használat során nem fogjuk őket összekeverni.

Például a `fioopen(TESZT);` biztosan a teszthez tartozó CFIO struktúrát fogja megnyitni.

3. Alkalmazási lehetőségek

3.1. Kezelhető feladattípusok

A file-kezelő alkalmazását olyan feladatok megoldásánál tartjuk célszerűnek, ahol az állományok nagyok, bonyolult a kulcsstruktúra felépítése, sokszor és sokféle szempont szerint kell a rekordokat elérni. Jól használható sok állomány egy programból történő kezelésére is. Listázó programok készítésénél a kontrollváltásos szerkezeteket jól támogatja.

Elsősorban adatfeldolgozó programrendszerek készítéséhez ajánljuk használatát. Pl:

- raktárnyilvántartás,
- pénzügyi rendszerek,
- számlázó programok,
- könyvelési programok stb.

3.2. Kezelhető állományszerkezetek

Azoknak az állományoknak a kezelését teszi lehetővé, amelyekben a rekordok hossza rögzített, a kulcsstruktúra minden eleme vagy növekvő, vagy csökkenő rendezettségű. A kulcs bárhol elhelyezkedhet a rekordban, legfeljebb 20 külön részben.

3.3. Más rendszerek által készített állományok gyors kezelése

A rendszer alkalmas más programok által készített állományok gyors rendezésére és feldolgozására is.

A szükséges fizikai paraméterekkel (fix rekordhossz, EOF jelet nem tartalmaz, definiálható kulcsstruktúra) rendelkező állományok egy build eljárással és egy ciklusban a next eljárás segítségével a kívánt sorrendben gyorsan végig olvashatóak.

Természetesen más eljárások segítségével is kezelhetjük ezeket az állományokat, például a pack eljárással rendezhetjük az állomány rekordjait.

4. A rendszer szolgáltatásai

4.1. Beépített hibakövető rendszer

A CFIO rendszer a tesztelés megkönnyítésére beépített hibanyomkövető rendszerrel rendelkezik, ami tetszés szerint ki- és bekapcsolható a 'fioerror' rendszerváltozó tartalmának megfelelően. Ha ennek értéke 1, akkor a hibafigyelő rendszer működik.

Bekapcsolt fioerror esetén, ha az eljárások végrehajtása során a rendszer hibát észlel, akkor a képernyő közepén alul egy villogó ablakban megjelenik a hibaüzenet. A hiba szövegének kiírását hangjelzés kíséri. A szöveg egy tetszőleges billentyű lenyomásáig látható. Ez a kiírás a programképernyőket nem rontja el.

A CFIO hibaszövegei angol nyelvűek, amelyek pontos felsorolása a 8. fejezetben található. Hasonló technikával dolgoznak a utility programok és a demo program is, de itt a programok futása során a fioerror mindig bekapcsolt állapotban van, és a programok saját üzenetei magyar nyelvűek.

Ha a fioerror bekapcsolt állapotban van, akkor a build függvény működése közben a képernyő jobb felső sarkában egy ablak mutatja, hogy hol tart az indexállomány építése. A beolvasási fázisban a READING KEYS felirat látható. Az indexállomány generálási szakaszában az éppen kiírás alatt lévő lap sorszámát láthatjuk a WRITE PAGE felirat után.

Bekapcsolt fioerror mellett a pack függvény működése közben a képernyő jobb felső sarkában látható, hogy hol tart az állomány tömörítése. A beolvasási fázisban a READING PAGE felirat látható az éppen munkában lévő lap sorszáma előtt. Az output készítési szakaszban az éppen kiírás alatt lévő lap sorszámát láthatjuk a WRITE PAGE felirat után.

Természetesen a build és a pack eljárás ablakos kiírásai sem rontják el a programképernyőt. Az eljárás működésének befejezése után az ablak alatti képernyőterület eredeti tartalma lesz látható.

4.2. Az indexfile felépítésének vizsgálata

A rendszer szolgáltatásait bővíti az indexfile felépítését bemutató DUMP nevű utility program. A program csak az indexállomány felépítését vizsgálja, és nem is kell tudnia, hogy az indexfile melyik adatállományhoz tartozik. A program hívásakor argumentumként kell megadni a vizsgálandó indexfile nevét.

Részletes leírása a 6.1. pontban található.

4.3. Sérült indexállományok helyreállítása

Ezt a feladatot a build nevű utility program látja el, amely egy állomány indexfile-jának a felépítését végzi. A paramétereket párbeszédés formában adhatjuk meg a képernyőről. Ezzel a programmal lehet egy más módon készült, de fix hosszúságú rekordokat tartalmazó adatfile-t a CFIO rendszer függvényei által kezelhetővé tenni, vagy egy sérült indexfile-t könnyen ujjaépíteni.

Az indexállomány felépítése elvégezhető mind a build utility program segítségével közvetlenül, mind a build CFIO eljárás segítségével a felhasználói programból.

Részletes leírás a 6.2. pontban található.

4.4. A sok üres rekordot tartalmazó állományok tömörítése

Ezt a feladatot a pack nevű utility program látja el, amely egy állomány átmásolását végzi kulcs szerint rendezett sorrendben. A másolás és az új indexállomány felépítése során a program az üres rekordokat kihagyja az állományból. A paramétereket párbeszédés formában adhatjuk meg a billentyűzetről. A program segítségével a tömörebb, rendezett állományokon végzett feldolgozás futási ideje csökken. Mindenképpen javasoljuk a sok rekordmozgatást igénylő (törlés-beszúrás), de általában egyféle indexállománnyal használt CFIO állományok időnkénti tömörítését. A pack eljárás működéséhez rendelkezésre kell állnia a karbantartott indexállománynak is.

A tömörítés elvégezhető mind a pack utility program segítségével közvetlenül, mind a pack CFIO eljárás segítségével a felhasználói programból.

Részletes leírás a 6.3. pontban található.

5. A rendszerben használható eljárások és a CFIO.H header file

5.1. A CFIO.H és a rendszerben használt változók

A rendszerlemezen található egy CFIO.H header file, amelyet minden esetben be kell illesztenünk a CFIO-t használó programba. Ez tartalmazza a struktúra definícióját, valamint a definiálandó külső változókat. A fiopage, fionumber, fiologpage, fioerror és a *fkulcs olyan rendszerváltozók, amelyeket a felhasználó is állíthat, vagy állítania is kell. Arról majd a későbbiek során lesz szó, hogy mi módon. A fiofn és fioin tömbök az adatfile-ok és az indexfile-ok handler-jeit tartalmazzák. Ezeket átírni tilos.

A file-ok száma a MAXFILE átirásával a szükségesnek megfelelően módosítható.

A kulcsok rögzített darabszámát megváltoztatni nem szabad.

A struct fileio típusú fkl tömb tartalmazza a cfio típusú file-ok deklarációit. Ezt a tömböt legegyszerűbb kezdeti értékadással feltölteni. Ld. 2.3. pontban a mintát.

```
/* CFIO.H          header file a CFIO allomanyok hasznalatahoz
```

```
                MAGISTER SOFTWARE    1987
```

```
                (c)  Marjai GY. & Marjai T.*/
```

```
#define MAXFILE 20
struct key          /* kulcsjellemezok struktura */
{
    int keypos;      /* kulcsdarab kezdopont */
    int keylength;   /* kulcsdarab hossz */
};
struct fileio       /* file-leiro struktura */
{
```



```

char *filename; /* adatfile nevenek a mutatoja */
char *indexname; /* indexfile nevenek a mutatoja */
int order;      /* novekvő=1, csökkenő=0 */
int length;     /* rekordhossz */
int keynum;     /* kulcsdarabok száma */
struct key keypieces[20]; /* a kulcsleíró struktúra */
}
fkl[MAXFILE];
int fiofn[MAXFILE], fioin[MAXFILE];
int fiopage, fionumber, fiologpage, fioerror;
struct fileio *fkulcs;

```

A következő fejezetekben az egyes könyvtári file-kezelő függvények leírása következik.

A leírások szerkezete a következő:

Összefoglalás

Az eljárás nevét, és paramétereit itt definiáljuk.

Leírás

Az eljárás feladatát, valamint használatának legfontosabb szabályait definiáljuk ebben a részben.

Visszatérési értékek

Itt az eljárások visszatérési értékeit definiáljuk, a kód jelentésével együtt.

Lásd még

Ebben a pontban az eljáráshoz szorosan kapcsolódó egyéb eljárásokat, file-okat soroljuk fel.

Példa az eljárás hívására

Ebben a pontban az eljárás leggyakoribb használatának egy mini mintaprogramját mutatjuk be.

5.2. FCREAT

Összefoglalás

```
int fcreat(file)           a file-struktúra sorszáma
int file;
```

Leírás

Létrehoz egy egyetlen üres rekordot tartalmazó adatállományt és egy ehhez tartozó indexfile-t a file-struktúra leírásnak megfelelően.

Használatára akkor van szükség, ha egy CFIO állomány eddig még nem létezett és most akarjuk létrehozni.

Visszatérési értékek

2 = már létezett ilyen indexfile
1 = már létezett ilyen adatfile
0 = sikerült létrehozni az üres állományt

Lásd még

fioopen()

Példa

```
int m, file;
.
.
file=0;
.
.
m=fcreat(file);
if(m==0)
    printf("Létrehoztam az fkl[0]-nak megfelelo allomanyt!");
else
    printf("Nem hozhato létre! Hibakod=%d",m);
.
.
```


5.3. FIOOPEN

Összefoglalás

```
int fioopen(file)
int file;          a file-struktúra sorszáma
```

Leírás

A 'file' értékének megfelelő sorszámu struktúra adat- és indexállományát megnyitja. A file-ok handler-jeit a fiofn, fioin tömbök 'file'-edik elemébe tölti.

Használatára akkor van szükség, ha valamely CFIO struktúrát az eljárások nyitott állapotban igényelnek.

Visszatérési értékek

```
0 = sikeres nyitás
-1 = nem létezik ilyen indexfile
-2 = nem létezik ilyen adatfile
```

Lásd még

```
fcreat(), search(), insert(), delete(), next()
```

Példa

```
int m,file;
.
.
file=0;
.
.
.
m=fioopen(file);
if (m<0)
{
    m=fcreat(file);
    if(m>0)
```



```

        {
            printf("A file nem nyithato!");
            exit(1);
        }
        m=fioopen(file);
    }
    printf("Sikeres file-nyitas!");
    .
    .

```

5.4. FIOCLOSE

Összefoglalás

```

void fioclose(file)
int file;           a file-struktúra sorszáma

```

Leírás

A nyitott file-struktúra lezárását végzi.

Használatára akkor van szükség, ha a file-struktúrát már nem akarjuk használni, vagy egy olyan eljárással akarunk dolgozni, amely lezárt állományon dolgozik.

Visszatérési érték

nincs

Lásd még

fioopen()

Példa

```

int m, file;
.
.

```



```

.
file=0;
.
fioclose(file);
.
.

```

5.5. SEARCH

Összefoglalás

```

long search(file, key, buff)
int file;           a file-struktúra sorszáma
char *key;          a keresett kulcs mutatója
char *buff;         a megtalált rekord mutatója

```

Leírás

Az eljárás az állomány rekordjai közül kiválasztja a keresett kulcsértéknek megfelelőt, ha pedig ilyen nincs, akkor a nála éppen nagyobbát. A file-struktúrának nyitott állapotúnak kell lennie a hívás előtt. A megtalált rekordot leíró értékeket (lapsorszám, rekordsorszám, a lap logikai sorszáma) a rendszerváltozókba helyezi el.

Az eljárás a megtalált rekordot a buff-ba tölti. Ha a keresett kulccsal megegyező rekordot talál, akkor azt, ha ilyen nincs, akkor a nála nagyobb kulcsúak közül az elsőt. Ha nincs nála nagyobb kulcsú rekord, akkor a rekordpuffer a rekordhossznak megfelelő számú hexa 7C karaktert (függőleges vonal) fog tartalmazni.

Használatára szükség van

- ha keresünk egy bizonyos kulcsú rekordot,
- INSERT függvény használata előtt,
- NEXT használata előtt, ha egy bizonyos kulcstól akarunk olvasni,
- DELETE használata előtt.

Ha megtalálja az adott kulcsú rekordot, akkor a rekord fizikai helyének megfelelő értékkel tér vissza az eljárás.

Az eljárás használata előtt a CFIO állományt fioopen-nel meg kell nyitni!

Visszatérési értékek

poz. = a megtalált rekord fizikai sorszáma az adatfile-ban
0 = nincs ilyen kulcsú rekord az állományban
-1 = file-definíciós vagy file-kezelési hiba

Lásd még

fioopen(), insert(), delete(), next(), <cfio.h>

Példa

```
int file;
long l;
char buffs[100];    /* a rekord merete legfeljebb 99 byte */
.
.
file=0;
.
fioopen(file);
.
.
l=search(file,"kulcs",buffs);
if(l<0)
{
    printf("search hiba!");
    exit(1);
}
if(l>0)
    printf("Volt ilyen rekord : %s",buffs);
else
    printf("Nem volt ilyen kulcsu rekord!");
.
.
fioclose(file);
.
.
```


5.6. INSERT

Összefoglalás

```
int insert(file, buff)
int file;           a file-struktúra sorszáma
char *buff;         a befűzendő rekord mutatója
```

Leírás

Az insert rutin az előzőleg meghívott search rutin által meghatározott logikai helyre fűzi be a buff-ban található rekordot. Ezt az információt a rendszerparaméterekbe helyezte el a search. Éppen ezért vigyázni kell arra, hogy a paraméterek értékei a két rutin hívása között ne változzanak meg. Ha szükséges, akkor a fiopage, fionumber, fiologpage értékét menteni kell.

Ha a kiválasztott lapon már nincs hely (a tartalma elérte az 1022 rekordot), akkor az insert rutin lapfelezéssel csinál helyet az indexállományban az új rekord címének. Ha túl sok éppen felezett oldalunk van, akkor célszerű az indexállományt build-del újjá építeni, vagy pack-kal tömöríteni.

Az insert eljárás, ha van az adatállományban törölt rekord, akkor fizikailag annak helyére illeszti, ha nincs, akkor a file végére helyezi az újat.

Használatára akkor van szükség, ha egy új rekordot akarunk az állományba illeszteni.

Visszatérési értékek

- 0 = sikeres beszúrás
- 1 = a file tele van, legfeljebb 65024 rekordja lehet
- 2 = hibás lapsorszám
- 3 = hibás rekordsorszám

Lásd még

fiopen(), search(), fioclose(), <cfio.h>

Példa

```
int m, file;
long l;           /* search visszateresi ertek */
char buffs[100];  /* ebbe helyezem a beszurni kivant rekordot */
char buff[100];   /* munkaterulet a searchnek */
char kulcs[20];   /* ebbe valogatom search előtt a kulcsot */
.
.
file=0;
.
fiopen(file);
l=search(file,kulcs,buff);
if(l>0)
    printf("Van mar ilyen kulcsu rekord!");
if(l==0)
{
    m=insert(file,buffs);
    if(m==0)
        printf("Sikeres beszuras!");
    else
        printf("Insert hiba! Hibakod=%d",m);
}
.
.
.
fioclose(file);
.
.
```

5.7. DELETE

Összefoglalás

```
int delete(file,buff)
int file;           a file-struktúra sorszáma
char *buff;         a törölt rekord mutatója
```


Leírás

A delete eljárás segítségével, a search rutinnal beállított helyről törölhetünk egy rekordot. Hasonlóképpen az inserthez, a delete sem használható önmagában, csak search-el együtt. Ha a két eljárás hívása között használunk olyan más eljárást, amely a rendszerparamétereket felülírja, akkor ezeket közben menteni kell.

Az eljárás a törlendő rekordot kimenti a buff nevű pufferba. Az indexállományból törli a rekord címét a megfelelő lapról, és a rekord címét beírja az üres címek nyilvántartásába. Az adatállományban a rekordot feltölti hexa 7C (függőleges vonal) karakterekkel.

Abban a speciális esetben, ha egy lapról az utolsó rekordot töröljük, akkor az eljárás automatikusan újra építi az indexfile-t.

A CFIO állományt az eljárás használata előtt fioopen-nel meg kell nyitni!

Használatára akkor van szükség, ha rekordot akarunk törölni az állományból és a rekordot search eljárással jelöltük ki.

Visszaterési értékek

- 0 = sikeres törlés
- 1 = hibás lapsorszám
- 2 = hibás rekordsorszám
- 3 = nincs elég memória a puffereknek
- 4 = nem lehet felépíteni az új indexfile-t

Lásd még

fioopen(), search(), fioclose(), <cfio.h>

Példa

```
int m, file;
long l;                /* search visszateresi ertek */
char buffs[100];       /* ide kerul majd a torolt rekord */
```



```

char buff[100];          /* munkaterület a searchnek */
char kulcs[20];          /* ide helyezzük search előtt a kulcsot */
.
.
file=0;
.
fiopen(file);
l=search(file,kulcs,buff);
if(l==0)
    printf("Nincs ilyen kulcsu rekord!");
if(l>0)
{
    m=delete(file,buffs);
    if(m==0)
        printf("Sikeres torles! A torolt rekord=%s",buffs);
    else
        printf("Delete hiba! Hibakod=%d",m);
}
.
.
.
fioclose(file);
.

```

5.8. NEXT

Összefoglalás

```

int next(file,buff)
int file;          a file-struktúra sorszáma
char *buff;        a rekordpuffer mutatója

```

Leírás

A next eljárás a CFIO file rendszerparaméterek által definiált rekordját a buff-ba tölti és a rendszerparamétereket a következő rekordra állítja.

Az első next hívás előtt be kell állítani a rendszerparamétereket a belépési pontra. Ezt megtehetjük

értékadással, vagy a search rutin elvégzi helyettünk úgy, hogy egy bizonyos kulcsú rekordra, vagy ha ilyen nincs, akkor a logikai sorrendben következőre állítja a rendszerparamétereket. A file elejére állításhoz például a fiopage=1 (fizikai lapsorszám), fionumber=1 (rekordsorszám a lapon belül), fiologpage=1 (logikai lapsorszám) értékadásokra van szükség.

Ha van még következő rekord, a next az előbb felsorolt paramétereket a következőre állítja át. Ha elérte a file végét, akkor ezt a visszatérési értékben jelzi és a paramétereket 1-re, vagyis a file elejére állítja. Ne feledkezzünk meg arról, hogy amikor a next állományvéget jelez a visszatérési értékben, akkor még a pufferben található az állománynak az utolsó rekordja.

A CFIO állományt az eljárás használata előtt fioopen-nel meg kell nyitni!

Használatára akkor van szükség, ha egy állomány rekordjait, vagy egy részének rekordjait kulcs szerint rendezett sorrendben szeretnénk elérni.

Visszatérési értékek

- 1 = sikeres betöltés, ez volt az utolsó
- 0 = sikeres betöltés, van még következő rekord
- 1 = hibás lapsorszám
- 2 = hibás rekordsorszám

Lásd még

fioopen(), search(), fioclose(), <cfio.h>

Példa

```
int m, file;  
char buffs[100];  
.  
.
```



```

file=0;
fiocopen(file);
.
.
fiopage=1;      /* Az elejetol listazzuk a rekordokat. */
fionumber=1;
fiologpage=1;
while((m=next(file, buffers)==0)
        printf("\nRekord=%s", buffers);
if(m>0)
        printf("\nUtolso=%s", buffers);
else
        printf("\nNext hiba. Hiba kod=%d", m);
.
.
fioclose(file);
.

```

5.9. BUILD

Összefoglalás

```

int build(file)
int file;          a file-struktúra sorszáma

```

Leírás

A build eljárás felépíti a CFIO file indexállományát. Az eljárás akkor működik, ha az adatfile létezik és az indexfile vagy nem létezik, vagy nem 'update'. Ha az indexfile-t újjá akarjuk építeni és az utolsó karbantartás ezzel az indexállománnyal történt, akkor a build előtt törölni kell az indexállományt. Az eljárás viszonylag tömör lapszerkezetet épít fel. Egy-egy lapot 90%-ig tölt fel, így a feltöltött állományok -- már csak eseti -- karbantartása lapmozgatás nélkül elvégezhető, és az indexállomány kellően tömör lesz.

A build a legmemóriaigényesebb eljárás a rendszerben. A

szükséges memóriaigény úgy számolható ki, hogy a kulcs hosszát megszorozzuk 256-al, ehhez hozzáadjuk a file rekordszámának a kétszeresét és még 2 Kbyte-ot.

Például egy 6000 rekordból álló 15 byte-os kulccsal rendelkező állomány build-jéhez $15 \times 256 + 6000 \times 2 + 2048 = 17888$ byte memóriaterületre van szükség. Természetesen ez a kb. 18 Kbyte memóriaigény csak a szükséges szabad adatterületet jelenti. Ennek nem a 64 kbytes default szegmensben kell rendelkezésre állnia, hanem bárhol lehet a memóriában.

Különösen nagyméretű állományok build-elése esetén javasoljuk a fioerror 1-re állítását az eljárás hívása előtt, mert akkor a képernyő jobb felső sarkában követhetjük, hogy hol tart az indexállomány építése anélkül, hogy a programképernyőt elrontanánk.

Az eljárás használata előtt nem szabad erre az állományra fioopen-t használni.

Használatára akkor van szükség, ha egy indexállomány még nem létezik, vagy megsérült, vagy pedig nem 'update'.

Visszatérési értékek

- 1 = az indexfile 'update'
- 0 = sikeres indexépítés
- 1 = az adatfile nem létezik
- 2 = üres az adatfile
- 3 = file-definíciós hiba, vagy a rekordok hossza nem fix
- 4 = túl nagy állomány, legfeljebb 65024 rekordja lehet
- 5 = nincs elég memória a file-szerkezet felépítéséhez
- 6 = nem lehet létrehozni az új indexfile-t
- 7 = nincs elég memória a munkaterületek számára
- 8 = nincs elég memória a lapterület számára

Lásd még

pack(), <cfio.h>, BUILD utility (6.fejezet)

Példa

```
int m, file;
.
.
file=0;
.
.
m=build(file);
if(m==0)
    printf("Az indexfile elkészült!");
if(m<0)
    printf("Hiba a buildben! Hibakod=%d",m);
.
.
.
/* Ha semmilyen kiírás sem jelenik meg, akkor
az indexfile-t nem kell újra építeni, mert 'update' */
.
```

5.10. PACK

Összefoglalás

```
int pack(file)           a file-struktúra sorszáma
int file;
```

Leírás

Az eljárás a már nagyon sok üres rekordot tartalmazó állományokból az üres rekordok kiemelését végzi el. Ez az egyetlen eljárás, amely az adatfile rekordjait az eredeti helyükről elmozgatja. A pack által létrehozott állomány a pillanatnyi állapotnak megfelelően a kulcs szerint sorbarendezve tartalmazza a rekordokat és nem tartalmaz üres rekordot.

Az eljárás használata feltételezi, hogy az indexállomány

rendelkezésre áll és 'update'. Ha ez nem áll fenn, akkor előbb a build eljárást kell használni.

Ez az egyetlen olyan eljárás, amely az adatfile átmásolásához munkaterületet igényel a kijelölt lemezmeghajtón. Gondoskodni kell róla, hogy az adatfile méretének megfelelő munkaterület rendelkezésre álljon. Az eljárás mind az indexállományt, mind pedig az adatfile-t, tehát az egész CFIO struktúrát átrendezi.

Az eljárás használata előtt nem szabad erre az állományra fioopen utasítást kiadni!

Használatára akkor van szükség, ha az állomány már túl sok üres rekordot tartalmaz, és emiatt a kezelése lelassul. Ha bármilyen ok miatt a fizikailag rendezett adatállományra van szükségünk, akkor is ezt az eljárást kell használni.

Visszatérési értékek

- 0 = sikeres file-tömörítés
- 1 = nem létezik a tömörítendő CFIO struktúra
- 2 = üres az állomány
- 3 = a munkaállomány a kijelölt meghajtón nem nyitható meg
- 4 = nincs elég memória a munkaterületek számára
- 5 = az új indexfile nem nyitható meg

Lásd még

build(), <cfio.h>, PACK utility (6.fejezet)

Példa

```
int m, file;  
.  
.  
file=0;  
.  
.  
m=pack(file);  
if(m==0)
```



```

        printf("Sikeres allomanytomorites!");
else
        printf("Tomoritesi hiba! Hibakod=%d",m);
.
.

```

5.11. REREAD

Összefoglalás

```

int reread(file,buff,rnumber)
int file;           a file-struktúra sorszáma
char *buff;         a kiolvasott rekord mutatója
long rnumber;       a kiolvasandó rekord fizikai sorszáma

```

Leírás

Az eljárás az adatfile-ből az rnumber-edik rekord tartalmát tölti a buff-ba. Az eljárás akkor is működik, ha az indexfile nem létezik, vagy hibás a tartalma.

Az eljárás használata előtt nem szabad a CFIO file-t fioopen-nel megnyitni. Ha nyitva volt, akkor fioclose-zal le kell zárni.

Használatára akkor van szükség, ha egy ismert sorszámu rekordot akarunk az adatfile-ből kiolvasni.

Visszatérési értékek

```

0 = sikeres olvasás
-1 = nem létezik az adatfile
-2 = nincs ilyen rekord
-3 = pozicionálási hiba

```

Lásd még

fioopen(), rewrite()

Példa

```
int m, file;
long rnumb;           /* a kiolvasando rekord sorszama */
char buff[100];       /* ide kerul a rekord */
.
.
file=0;
.
rnumb=517;
.
m=reread(file, buff, rnumb);
if(m==0)
    printf("Sikeres olvasas! Rekord=%s", buff);
else
    printf("Hiba a reread-nel! Hibakod=%d", m);
.
.
```

5.12. REWRITE

Összefoglalás

int rewrite(file, buff, rnumber)	
int file;	a file-struktúra sorszama
char *buff;	a rekordpuffer mutatója
long rnumber;	a felülírandó rekord sorszama

Leírás

Az eljárás az adatfile rnumber-edik rekordjának tartalmát írja felül a buff tartalmával. A rekord felülírás előtti tartalma a buff-ba kerül vissza. A felülírást csak akkor hajtja végre az eljárás, ha a kulcsmező az új rekordban is változatlanul marad, így az eljárás használata nem rontja el az indexállományt.

Az eljárás akkor is működik, ha az indexfile nem létezik, vagy hibás a tartalma.

Az eljárás használata előtt nem szabad a CFIO file-t fioopen-nel megnyitni. Ha nyitva volt, akkor fioclose-zal le kell zárni.

Használatára akkor van szükség, ha egy ismert fizikai sorszámu rekordot akarunk az adatfile-ban felülírni.

Visszatérési értékek

- 0 = sikeres felülírás
- 1 = nem létezik az adatfile
- 2 = nincs ilyen rekord
- 3 = pozicionálási hiba
- 4 = más a kulcs, ezzel nem lehet felülírni a rekordot

Lásd még

fioopen(), search(), reread()

Példa

```
int m, file;
long rnumb;           /* a felulirando rekord sorszama */
char buffs[100];      /* rekordpuffer */
.
.
file=0;
.
.
fioopen(file);
rnumb=search(file, "kulcs", buffs);
fioclose(file);
if(rnumb<=0)
{
    printf("Nincs ilyen kulcsu rekord!");
    exit(1);
}
.
/* aktualizaljuk a buffs-ban levo rekordot */
.
```



```

m=rewrite(file,buffs,rnumb);
if(m==0)
    printf("Sikeres feluliras! A regi rekord=%s",buffs);
else
    printf("Hiba a rewrite-nal! Hibakod=%d",m);
.
.
.
.

```

5.13. KULCSVAL

Összefoglalás

```

void kulcsval(file,key,buff)
int file;           a file-struktúra sorszáma
char *key;          a kulcs első karakterére mutat
char *buff;         a rekordpuffer mutatója

```

Leírás

Az eljárás a file-struktúrában található kulcsstruktúra definíciójának megfelelően összeválogatja a rekordból a rekord kulcsát és key-be helyezi.

Az eljárás végrehajtásához természetesen nem kell magát a CFIO file-t megnyitni, de ha az nyitott állapotban van, az eljárást az sem zavarja.

Használatára akkor van szükség, ha van összeállított rekordunk és szükségünk van a rekord kulcsára.

Visszatérési érték

nincs

Lásd még

<cfio.h>

Példa

```
int m, file;
char key[20];
char buffs[100];
.
.
file=0;
.
.
/* buff-ba összeallitottunk egy rekordot */
.
kulcsval(file, key, buff); /* key-be kerül a rekord kulcsa */
.
fioopen(file);
.
/* megvizsgáljuk, hogy volt-e már ilyen rekord */
l=search(file, key, buffs);
fioclose(file);
.
.
```

5.14. ERRORW

Összefoglalás

void errorw(msg)
char *msg; az üzenet, ami az ablakban megjelenik

Leírás

Az eljárás nem tartozik szorosan a CFIO rendszerhez, de mivel a rendszer összes eljárása használja, ezért a felhasználónak lehetősége van arra, hogy a saját programjában is használja.

Az eljárás segítségével a paraméterként megadott üzenet a képernyő aljának közepére íródik. A kinyitott ablak alatti terület nem törlődik, tehát a program képernyője az ablak becsukása után változatlanul megmarad.

A szöveg kiírását hangjelzés kíséri. A villogó ablak a következő billentyű lenyomásáig marad a képernyőn.

Ha a képernyő ablak alatti területének mentéséhez nincs elegendő memória, akkor a program futása fatális hibajelzéssel (Not enough memory!) megszakad.

Visszatérési értékek

nincs

Lásd még

<cfio.h>(fioerror)

Példa

```
char *msg[2];  
.  
msg[0]=" Hibajelzes! ";  
msg[1]=" Uzenetek! ";  
.  
errorw(msg[0]);  
errorw(msg[1]);  
.  
.
```


6. Utility programok

6.1. DUMP

A DUMP.EXE utility program a CFIO felépítésű file-struktúrák indexállományának az ellenőrzésére, illetve vizsgálatára szolgál. Segítségével a képernyőre írhatjuk a CFIO állomány legfontosabb jellemzőit.

Ezek a következők:

- az állományt alkotó rekordok száma,
- az állományban található élő rekordok száma,
- az indexfile-t felépítő lapok száma,
- egy rekord kulcsmezőjének a hossza,
- a lapok logikai sorrendje
- az egyes lapokon található legelső kulcs.

A program hívásánál a vizsgálandó indexfile nevét argumentumként meg kell adni!

Példa a utility használatára

dump teszt.ind

Ezt a utility programot elsősorban a fejlesztés idején érdemes használni, de segíts'gével üzemeltetés közben is kiderülhet az indexfile sérülése. Segítségével egy 'gyanús' állományról eldönthetjük, hogy az CFIO szerkezetű indexfile-e vagy sem.

A program egy oldalra 13 lap adatait listázza, és utána egy

tetszőleges billentyű megnyomására hozza be a következő oldal adatait.

A program hibajelzései:

"Használja így: dump file-nev!"

- Híváskor nem adta meg az
indexfile nevét argumentumként.

"Ez a file nem létezik!"

- Nem létező file-t akart elemezni.

"Ez a file üres !"

- A file nem tartalmaz rekordot.

"Ez a file nem CFIO indexfile!"

- Nem CFIO szerkezetű az elemezni kívánt. file.

6.2. BUILD

A BUILD.EXE utility program a build eljárást általános formában használó, önállóan működő rendszerprogram. Segítségével megoldható a sérült vagy elveszett indexfile-ok gyors helyreállítása. Ennek a utility programnak a segítségével illeszthetünk hozzá egy más programrendszer által készített file-t a CFIO eljárásokat használó felhasználói programunkhoz.

A program működése előtt ellenőrzi, hogy létezik-e az adatállomány. Ellenőrzi azt is, hogy létezett-e már az indexállomány. Ha létezett, akkor csak úgy hajlandó felülni, ha az indexállomány dátuma régebbi, mint az adatállományé. Ebből következik, hogy a látszólag 'update', de sérült indexállományt a build használata előtt le kell törölni a lemezzről. Ezt akkor is így kell csinálni, ha nem a utility programot, hanem a felhasználói programba beépíthető eljárást használjuk.

A program által felépített indexállomány nem pontosan ugyanolyan, mint amilyen a karbantartás során automatikusan jön létre. Természetesen az indexeket ugyanolyan sorrendben tartalmazza, mint a karbantartás során létrejött állomány, de a lapokat egyenletesen 900 db rekordcímmel tölti fel. A maradék kerül az utolsó lapra. Jól látható a különbség, ha egy karbantartott indexállományt és egy build-del felépített indexállományt vizsgálunk a dump utility segítségével.

A harmadik terület -- ahol szükség van az indexállományok felépítésére -- amikor egy másik indexfile (kulcsstruktúra) szerint karbantartott adatállományt egy újabb CFIO struktúrában, másik kulcs szerint rendezve szeretnénk végigolvasni, lekérdezni vagy listázni. Ne felejtsük el, hogy ilyenkor a CFIO struktúrát -- ha előtte változott az adatállomány -- aktualizálni kell az indexállomány újraépítésével!

A program párbeszédés formában működik. Egyszerre több állományt is újraépíthetünk a segítségével. A program hívásakor argumentumokat nem adhatunk meg. A szükséges CFIO állományleíró mezőket a program automatikusan kitölteti a kezelőjével. Ugyanazokat a jellemzőket kell megadni, mint amit

a CFIO.H file tárgyalásánál definiáltunk:

- az adatállomány nevét,
- az indexállomány nevét,
- az adatrekord hosszát,
- a rendezettséget /1=növekvő 0=csökkenő/,
- a kulcs darabjainak a számát,
- páronként a kulcsdarabok kezdőpozícióját, hosszát.

A rovatok kitöltése értelemszerű. Az utolsó kulcsdarab definiálása után a képernyő alján megjelenik a kérdés: ok?

Az 'I' billentyű megnyomására elkezdődik az új indexállomány felépítése. Egyébként a legelső mezőtől kezdve javíthatjuk a begépett jellemzőket.

Az indexállomány felújítása után a program újra egy állomány jellemzőit kezdi bekérni. Ha nincs több felépítendő állományunk, vagy a jellemzők beadása közben meggondoltuk magunkat, akkor -- ha egy mezőtartalmat 'F1'-gyel zárunk le -- a program futása befejeződik.

6.3. PACK

A PACK.EXE utility program a pack eljárást általános formában használó, önállóan működő rendszerprogram. Segítségével megoldható egy file tényleges fizikai rendezése, az üres rekordoknak az adatállományból való kiszűrése.

Ennek az utility programnak a segítségével rendezhetünk egy más programrendszer által készített file-t is, ha a file egyébként megfelel a CFIO struktúrájú adatállományokkal szemben támasztott követelményeknek (fix rekordhossz, nincs a végén EOF jel, ismert a kulcsstruktúra). Ebben az esetben azonban az állományhoz először építenünk kell egy aktuális indexállományt a build segítségével, és csak ezután lehet a pack eljárással rendezni.

A program a működésének megkezdése előtt ellenőrzi, hogy létezik-e az adatállomány és az indexfile. Ellenőrzi a megadott értékből a rekordhossz rögzítettségét is.

A program által felépített indexállomány pontosan ugyanolyan, mint amilyent a build eljárás hoz létre. Természetesen nem ugyanolyan sorrendben tartalmazza az indexeket, hiszen az adatállományt alkotó rekordok fizikai sorrendje most a rendezési kulcsnak megfelelő. Így az új indexfile első lapja az 1-900-ig tartó címetek fogja tartalmazni, a második a 901-1800-ig és így tovább.

Ha egy build-del felépített indexállományt vizsgálunk a dump utility segítségével, és ugyanezt a pack utility használata után is megteesszük, akkor csak abban az esetben találunk

különbséget, ha a pack előtt az állomány még tartalmazott üres rekordokat. Természetesen a címek mások az átrendezés miatt, de ez a dump képernyőn nem látható.

Ne felejtsük el, hogy a pack előtt a CFIO struktúrát -- ha változott az adatállomány -- aktualizálni kell az indexállomány újraépítésével!

A program párbeszédés formában működik. Egyszerre több állományt is tömöríthetünk a segítségével. A program hívásakor argumentumokat nem adhatunk meg. A szükséges CFIO állomángleíró mezőket a program automatikusan kitölteti a kezelőjével. Ugyanazokat a jellemzőket kell megadni, mint amit a CFIO.H file tárgyalásánál definiáltunk, a rendezettség-mező kivételével. Erre most nincs szükség, hiszen a pack az 'update' indexállományban található sorrendben szedi össze a rekordokat, így itt nincs mód az eredeti rendezettség megváltoztatására. Ha mégis ilyen rendezést szeretnénk csinálni vagy a kulcsokat változtatni, akkor előbb egy annak megfelelő indexállományt kell build-del építeni. Meg kell adni tehát:

- az adatállomány nevét,
- az indexállomány nevét,
- az adatrekord hosszát,
- a kulcs darabjainak a számát,
- páronként a kulcsdarabok kezdőpozícióját, hosszát.

A rovatok kitöltése értelemszerű. Az utolsó kulcsdarab definiálása után a képernyő alján megjelenik a kérdés: ok?

Az 'I' billentyű megnyomására elkezdődik az állomány tömörítése. Egyébként a legelső mezőtől kezdve javíthatjuk a begépelte jellemzőket.

A tömörítés befejezése után a program újra egy állomány jellemzőit kezdi bekérni. Ha nincs több tömörítendő állományunk vagy a jellemzők beadása közben meggondoltuk magunkat, akkor -- ha egy mezőtartalmat 'F1'-gyel zárunk le -- a program futása befejeződik.

7. A CFIO rendszer DEMO programja

7.1. A DEMO.EXE program

Ezt a programot azért kellett megírni, mert a CFIO rendszer kipróbálása egyébként -- programírás nélkül -- nem lehetséges. A CFIO szoftver egy programkönyvtár, amely természetesen nem tartalmaz önállóan végrehajtható programot, így a rendszer működését bemutató demo rész sem kerülhetett ebbe.

Szeretnénk, ha mindenki -- már a demo program kipróbálása után -- képet tudna alkotni a CFIO rendszer lehetőségeiről.

A demo program összes CFIO eljárását a cfio.lib-ből a lefordított demo.c-hez szerkesztve használja, ahogy ez a forrásnyelvi listából is kiderül. A könyvtár egy részét (build, pack eljárások) a program nem használja, hiszen ezek egy adatfile birtokában a utility programok segítségével azonnal kipróbálhatók.

A demo programban megvalósított 'mini' adatkezelő rendszeren tanulmányozhatjuk a file-kezelés lehetőségeit. Meg akartuk kímélni a program használóját az adatbevitel fáradságos munkájától. Csak a feltétlenül szükséges adatokat kérjük be a billentyűzetről, a többit a MicroSoft C véletlenszám-generátora, a rand() függvény szolgáltatja.

A véletlenszám-generátor kezdőértékét nem állítjuk át. Azzal a szándékkal tesszük ezt, hogy megkönnyítsük a tesztelés során az azonos 'szituációk' előállítását a különböző akciók lebonyolításához. Ezzel biztosítottuk azt, ha valaki a demo programot ugyanolyan file-paraméterekkel újra indítja, akkor pontosan ugyanazt a tesztállományt tudja ismét előállítani. Más file-paraméterek megadása esetén (rekordhossz és rekordszám) természetesen a tesztfile különbözni fog az előzőtől, ezért a program elején az előző futásban generált teszcf.dat és tesztcf.cfk állományokat töröljük.

A demo programban a CFIO állományt leíró paramétereket némileg lehatároltuk, egyrészt a képernyős megjelenítés technikai korlátai, másrészt a mintaállományt előállító véletlenszám-generátor lassúsága miatt. A CFIO rendszer tényleges korlátai a dokumentáció 1. fejezetében találhatók.

A demo program által kezelt CFIO állomány legfeljebb 99 db, maximum 60 byte hosszúságú rekordot tartalmazhat. A program csak egyetlen CFIO állományt használ, persze nincs gyakorlati akadálya a felhasználói programokban akár 20 állományt is egyszerre nyitva tartani, de ügyelni kell rá, hogy a szükséges számú handler rendelkezésre álljon a DOS szintjén is.

A demo program forrásnyelvű szövege a 7.3. alfejezet-ben és a rendszerlemezen megtalálható. Ennek alapján más CFIO állományokat használó adatfeldolgozó feladatok is könnyen programozhatók. Ha valaki nincs megelégedve a demo program lehetőségeivel, akkor nyugodtan átírhatja a forrást, hiszen a cfio könyvtárba van csak szüksége a programszerkesztéshez a forrásnyelvű eljárásokon kívül. Így kisebb átalakítások könnyen elvégezhetők a MicroSoft C 4.00 programozási környezet segítségével.

Most pedig tekintsük végig a demo programot lépésről lépésre!

7.2. A DEMO működése

A bemutatkozó szövegek után a felhasználónak meg kell adnia egy CFIO struktúrájú file paramétereit.

A paraméterek:

- a rekord mérete byte-okban,
- növekvő/csökkenő rendezettség,
- kulcsdarabok száma,

- a kulcsdarabok kezdőpontjai,
- a kulcsdarabok mérete,
- a rekordok száma.

A paraméterek megadása után a program létrehozza az ennek megfelelő szerkezetű, CFIO rendszerű üres adatállományt. A véletlenszám-generátor segítségével bele is írja a megadott számú rekordot az állományba. Az újabb magyarázó szöveg után egy ablak nyílik a képernyőn, amelyből kiválaszthatjuk az éppen tesztelni kívánt CFIO rutint.

A rutinok:

SEARCH

Az eljárás bekéri a keresett rekord kulcsát a kulcsdefiniálásnak megfelelően. Azaz, ha például 2 kulcsdarab van és az első kezdőpontja 0, a hossza 3, a második kezdőpontja 5, hossza 4, akkor a kulcs hossza összesen 7 lesz, és a kulcsot rendre a rekord 0., 1., 2., 5., 6., 7., 8. byte-jai alkotják. A megfelelő hosszúságú kulcs begépelése után 4 eset lehetséges:

1. Olyan kulcsot adtunk meg, amelyhez létezik rekord. Ekkor a "keresett rekord" szöveg alatt visszakapjuk a teljes rekordunkat, és megkapjuk még a fizikai helyét is az adatfile-ban.
2. Olyan kulcsot adtunk meg, amelyhez nem létezik rekord, de van logikailag utána következő. Ekkor a "Nincs ilyen kulcsu rekord" üzenetet kapjuk, és utána képernyőre kerül a logikailag következő rekord tartalma.
3. Olyan kulcsot adtunk meg, amelyhez nem létezik rekord és nincs logikailag utána következő. Ekkor a "Nincs ilyen kulcsu rekord, ez a file vegere kerülne" üzenet jelenik meg. Ez azt jelenti, hogy az ilyen kulcsú rekord logikailag az utolsó után következne.

4. Valamilyen, angol nyelvű hibaüzenetet kapunk, amit már nem a demo program szolgáltat, hanem a CFIO rendszer. Ez valamilyen fatális hibára utal, ld. a 8. fejezet táblázatát.

INSERT

Az eljárás bekéri a beszúrandó rekord kulcsát. Lehetséges esetek:

1. Olyan kulcsot adtunk meg, amelyhez létezik rekord. Ekkor a "Van már ilyen kulcsu rekord" üzenetet kapjuk, és megjelenik a rekord is. A rekord nem kerül be a file-ba, mert a demo programban nem engedünk meg azonos kulcsú rekordokat.

2. Olyan kulcsot adtunk meg, amelyben szerepel legalább egy olyan byte, amelynek ASCII kódja nagyobb, mint 123. Ezt a rendszer nem engedélyezi a törölt rekordok felépítése miatt. A "Hibas a kulcs, túl nagy valamelyik ASCII kód" üzenetet kapjuk.

3. Szabályos, még nem létező kulcsot adtunk meg. Ekkor a program létrehozza a kulcshoz a rekordot (csak azért, hogy a felhasználót megkíméljük rekordnyi mennyiségű karakter begépelésétől), és beszúrja a file-ba. Ezután a "Beszurtam, nyomjon egy billentyűt !" üzenetet kapjuk.

4. Valamilyen, angol nyelvű hibaüzenetet kapunk, amit már nem a demo program szolgáltat, hanem a CFIO rendszer. Ez valamilyen fatális hibára utal, ld. a 8. fejezet táblázatát.

NEXT

Az eljárás kilistázza a két bekért kulcs által határolt rekordokat. Vigyázat, ha a rendezettség fordított (csökkenő), akkor a határokat is fordítva kell megadni! Lehetséges esetek:

1. Rosszul adtuk meg a határokat. Például a rendezettség csökkenő, és az alsó határ csupa egyes, a felső csupa kettes. Ekkor a "Hibas hatarok" üzenetet kapjuk.

2. A két határ közé nem esik rekord. Ekkor a "Nincs listazando tetel" üzenetet kapjuk.

3. Legalább egy rekord kulcsa a két határ közé esik. Ekkor megkapjuk az alsó határhoz logikailag legközelebb álló kulcsú rekordot, és a képernyő alsó sorában megjelenő menüből választhatunk:

Kovetkezo rekord -> ENTER EXIT -> F1 RUTINMENU -> F2

Ha nem szállunk ki a listázásból és már elfogytak a két határ közé eső rekordok, akkor a "Nincs tobb!" üzenetet kapjuk.

4. Valamilyen, angol nyelvű hibaüzenetet kapunk, amit már nem a demo program szolgáltat, hanem a CFIO rendszer. Ez valamilyen fatális hibára utal, ld. a 8. fejezet táblázatát.

REREAD

Az eljárás bekéri a beolvasandó rekord fizikai sorszámát. Lehetséges esetek:

1. Olyan fizikai sorszámot adtunk meg, amely nem létezik. Ekkor az "Invalid record number!" üzenetet kapjuk. Ebben a szituációban ez nem fatális hiba.

2. Létező fizikai rekordszámot adtunk meg. Ekkor megjelenik a képernyőn a keresett rekord.

3. Valamilyen, angol nyelvű hibaüzenetet kapunk (az "Invalid record number" kivételével), amit már nem a demo program szolgáltat, hanem a CFIO rendszer. Ez valamilyen fatális hibára utal, ld. a 8. fejezet táblázatát.

REWRITE

Az eljárás bekéri a felülírandó rekord fizikai sorszámát. Lehetséges esetek:

1. Olyan fizikai sorszámot adtunk meg, amely nem létezik. Ekkor az "Invalid record number!" üzenetet kapjuk. Ebben a szituációban ez nem fatális hiba.

2. Létező fizikai rekordszámot adtunk meg. Ekkor bekéri az új rekordot. Újabb 3 eset lehetséges:

2.a. Nem szabályos a rekord, azaz valamelyik byte-jának ASCII kódja nagyobb, mint 123. Ekkor a "Hibas adat, túl nagy ASCII kód !" üzenetet kapjuk.

2.b. Szabályos a rekord, de a kulcsa nem egyezik meg a cserélendő rekord kulcsával. Ezt nem engedhetjük meg a rendezettség fenntartása miatt. Ekkor az "Invalid key! I can't rewrite!" üzenetet kapjuk. Ebben a szituációban ez nem fatális hiba.

2.c. Szabályos a rekord, egyforma a kulcs. Ebben az esetben a "Felulírtam, nyomjon egy billentyűt !" üzenetet kapjuk.

3. Valamilyen, angol nyelvű hibaüzenetet kapunk (az "Invalid key! I can't rewrite!" kivételével), amit már nem a demo program szolgáltat, hanem a CFIO rendszer. Ez valamilyen fatális hibára utal, ld. a 8. fejezet táblázatát.

DELETE

1. Olyan kulcsot adtunk meg, amelyhez nincs rekord. Ekkor a "Nincs ilyen kulcsú rekord" üzenetet kapjuk.

2. Szabálytalan kulcsot adunk meg. Ekkor a "Hibas adat, túl nagy ASCII kód !" üzenetet kapjuk.

3. Szabályos, létező kulcsot adtunk meg. Ebben az esetben

megjelenik a képernyőn a törölni kívánt rekord, és a "Toroltem, nyomjon egy billentyut" üzenetet kapjuk.

4. Valamilyen, angol nyelvű hibaüzenetet kapunk, amit már nem a demo program szolgáltat, hanem a CFIO rendszer. Ez valamilyen fatális hibára utal, ld. a 8. fejezet táblázatát.

Vezérlés

A képernyő alsó sorában mindig az élő funkcióbillentyű-menü látható:

F1-> EXIT - kilépés a demóból

F2 -> RUTINMENU - kilépés a rutinból

7.3. A DEMO program forrásnyelvi listája

```
/* DEMO.C */
```

```
/*=====*
*                                     *
*          CFIO DEMO PROGRAM          *
*                                     *
*      (c) 1987   Marjai Gyorgy & Marjai Tamas   *
*                                     *
*          M A G I S T E R   S O F T W A R E      *
*=====*/

#include <stdio.h>
#include <fcntl.h>          /* a hasznalt header file-ok */
#include <io.h>

#define ON 1                /* a definialando konstansok */
#define OFF 0
#define YES 1
#define IGEN 23
```



```

#define MAXFILE 1      /* most csak 1 CFIO file-t hasznalunk */
#define MAXKEYSIZE 20  /* kulcsdarabok maximalis szama */

                        /* szin byte beallitas */
#define COLOR(f,b) ((f & (7 << 4)) | ( (b&7) << 4))

#define CWINDOWFRAME COLOR((4|8),0) /* ablakszin beallitasok */
#define CWINDOWTEXT  COLOR(0,2)
#define CMMSGWIND  COLOR(0,(3|8))
#define CWINDOWMARK  112

#define ENTER      28      /* A szukseges billentyuk 'scan' kodjai */
#define F1          59
#define F2          60
#define F6          64
#define F7          65
#define F8          66
#define F9          67
#define F10         68

#define HOME       71
#define UP         72
#define PGUP       73
#define LEFT       75
#define RIGHT      77
#define END        79
#define DOWN       80
#define PGDN       81
typedef char far *WINDTYPE; /* az ablak tipusa */
WINDTYPE windsave();

struct key          /* kulcsjellemzok strukturaja */
{
    int keypos;
    int keylength;
};

struct fileio       /* file-leiro struktura */
{
    char *filename;
    char *indexname;
    int order;
};

```



```

        int length;
        int keynum;
        struct key keypieces[MAXKEYSIZE];
    }
    fkl[MAXFILE],*fkulcs;

char *ceg = "MAGISTER SOFTWARE";

char *s[] = {           /* a rutinmenu elemeinek szovegei */

    "SEARCH -> Kulcs szerinti kereses      ",
    "INSERT -> Kulcs szerinti befuzes      ",
    "NEXT   -> Sorrendben kovetkezo olvasas",
    "REREAD -> Fizikai szam szerinti olvasas",
    "REWRITE -> Fizikai szam szerinti iras  ",
    "DELETE -> Kulcs szerinti torles       ",
    "VEGE                                     "};

int cedit = 112;          /* kepernyo szovegszinek beallitasa */
int cetext = 112;
int cmenutext = COLOR(3,0);
int cmenumark = COLOR((6|8),0);
int cwritetext = COLOR(6,0);
int cwritescr = COLOR(2,0);
int ctitlescr = COLOR((5|8),0);

int fioin[MAXFILE];/* az indexfile-ok handler-jeit tartalmazza */
int fiofn[MAXFILE];/* az adatfile-ok handler-jeit tartalmazza */

int fiopage;              /* a fizikai lapszam */
int fionumber;            /* a lapon belül a cim sorszama*/
int fiologpage;           /* a logikai lapszam */
int fioerror = 1;         /* a hibakoveto bekapcsolva */

unsigned savecurs,curstype(); /* kurzorvisszaallitashoz */
WINDTYPE fct2;            /* ablak tipusu mutato */
unsigned rekszam;         /* rekordszam */
char *buff,*kulcs;        /* rekordpuffer, kulcspuffer mutatok */
int oszlopk,oszlopr;      /* kozeprehelyezeshez oszlopok */
int kulcshossz;           /* az osszerakott kulcsdarabok hossza */
int file = 0;             /* a CFIO struktura szama */
                           /* most csak a nullast hasznaljuk */

```



```

main()
{
    int i,j;                                /* ciklusváltozók */
    char *malloc();                         /* a tarfoglalás függvénye */
    char *buff1,*buff2;                    /* munkapufferek */
    long ve;                                /* visszatérési érték */
    int m,h,cs;
    int karakter;                            /* a véletlenszám-generator által
                                           létrehozott karakterérték */
    int melyiket;                           /* a rutinmenu melyik elemét használjuk */

    unlink("tesztcf.dat"); /* a régi tesztadatok letörése */
    unlink("tesztcf.cfk");
    fkl[0].filename="tesztcf.dat"; /* az adatfile neve */
    fkl[0].indexname="tesztcf.cfk";
    erase();                               /* képernyőtörés */
    savecurs=curstype(-1,-1);             /* kurzoreltüntetés */
    reklam();
    szoveg1();
    ablak1();
    szoveg2();
    filedef();
    msgwrite("Létrehozom a file-t -> FCREAT");
    fcreat(file);
    msgwrite("Létrehoztam! Most vszg-vel generalom a rekordokat!");
    fioopen(file);
    buff=malloc(fkl[0].length+1);
    buff1=malloc(fkl[0].length+1);
    buff2=malloc(fkl[0].length+1);
    m=1;
    for(i=0; i<fkl[0].keynum; i++)
        m+=fkl[0].keypieces[i].keylength;
    kulcs=malloc(m);
    kulcshossz=m-1;
    oszlopk=(80-kulcshossz)/2;
    oszlopr=(80-fkl[0].length)/2;
    for(i=0; i<rekszam; i++)
    {
        for(j=0; j<fkl[0].length; j++)
        {
            karakter=rand()/762+47;
            buff1[j]=karakter;
        }
    }
}

```



```

buff1[fkl[0].length]='\0';
kulcsval(file,kulcs,buff1);
ve=search(file,kulcs,buff2);
if(ve!=0)
    errorw("Volt már ilyen kulcsu rekord");
else
    insert(file,buff1);
}
szoveg3();
while(1)
{
    msgwrite("Valasszon az ablakbol !");
    melyiket=select(5,20,38,6);
    switch(melyiket)
    {
        case 0:
            keres();
            goto cveg;
        case 1:
            beszur();
            goto cveg;
        case 2:
            gorget();
            goto cveg;
        case 3:
            fizolv();
            goto cveg;
        case 4:
            fizir();
            goto cveg;
        case 5:
            torles();
            goto cveg;
        case 6:
            free(buff1);
            free(buff2);
            fioclose(file);
            erase();
            exit(0);
        cveg:;
            screen("RUTIN VALASZTAS");
            break;
    }
}

```



```

reklam()
{
WINDTYPE fc; /* ablak tipusu valtozo */
int bfs,bfo; /* az ablak bal felso sora,bal felso oszlopa */
int jas,jao; /* az ablak jobb also sora,jobb also oszlopa */
int h;
bfs=1;
bfo=0;
jao=20;
jas=5;

/* az ablak alatti terület elmentese*/
if((fc=winsave(bfs,bfo,jas,jao))==NULL)
{
erase(); /* nem sikerult */
printf("WINDSAVE ERROR (SELECT) !");
fioclose(file);
exit(0);
}

windfram(fc,1,MSGWIND); /* az ablak keretenek rajzolasa */
cupo(bfs+1,bfo+1); /* kurzorvezerles */
aputtys(" M A G I S T E R",MSGWIND); /* attributumos kiiras */
cupo(bfs+3,bfo+1);
aputtys(" SOFTWARE",MSGWIND);
h=0;
while(h!=-1)
h=windmove(fc,3); /* az ablak mozgatasa jobbra */
h=0;
while(h!=-1)
h=windmove(fc,2); /* lefele */
h=0;
while(h!=-1)
h=windmove(fc,1); /* balra */
h=0;
while(h!=-1)
h=windmove(fc,0); /* fel */
sound(1200);delay(800);nosound();/* hang */
}
szoveg1()
{
cupo(6,15);
aputtys("Koszontom Ont, a program felhasznalojat !",cwritescr);
cupo(8,20);

```



```

szoveg3()
{
char szov[4];
screen("A RENDSZER RUTINJAI");
itoa(szov, rekszam, 3);
cupo(6, 10);
aputtys("Letrehoztam egy ", cwritescr);
aputtys(szov, cwritescr);
aputtys("db. rekordbol allo ", cwritescr);
cupo(8, 10);
aputtys("tesztcf nevű file-t es a hozza tartozo", cwritescr);
cupo(10, 10);
aputtys("kulcsfile-t amely az eloirt rendezettsegnek", cwritescr);
cupo(12, 10);
aputtys("megfeleloen tartalmazza az adatfile rekordjainak"
        , cwritescr);
cupo(14, 10);
aputtys("cimeit. Elkezdhetjuk a rutinok kiprobalasat."
        , cwritescr);
cupo(16, 10);
msgwrite("Nyomjon egy billentyut");
kbflush();
kbget();
}
aputtys("Ismerkedjunk meg a", cwritescr);
cupo(11, 20);
aputtys("C", cwritetext);
aputtys(" - c nyelvű", cwritescr);
cupo(13, 20);
aputtys("F", cwritetext);
aputtys(" - file-kezelő", cwritescr);
cupo(15, 20);
aputtys("I", cwritetext);
aputtys(" - input", cwritescr);
cupo(17, 20);
aputtys("O", cwritetext);
aputtys(" - output", cwritescr);
cupo(20, 20);
aputtys("rendszer lehetosegeivel !", cwritescr);
cupo(24, 18);
aputtys("Nyomjon egy billentyut !", cwritetext);
delay(1000);

```



```

szoveg2()
{
screen("A RENDSZER LEIRASA");
cupo(6,10);
aputtys("A rendszer a lemezen tarolt adatok gyors kulcsszerinti"
,cwritescr);
cupo(8,10);
aputtys("elereset,modositasat,letrehozast segiti elo egyszeru"
,cwritescr);
cupo(10,10);
aputtys("fuggvenyek segitsegevel. (Az eppen hasznalt eljaras"
,cwritescr);
cupo(12,10);
aputtys("neve a kepernyo cim vagy uzenet soraban lathato.")
,cwritescr);
cupo(14,10);
aputtys("Eloszor is hozzuk létre a file-t, ehhez a kovetkezo",
cwritescr);
cupo(16,10);
aputtys("informaciokra van szuksegunk, amelyeket kerjuk, most",
cwritescr);
cupo(18,10);
aputtys(" On határozzon meg !",cwritescr);
msgwrite("Nyomjon egy billentyut");
kbflush(); /* billentyuzetpuffer torles */
kbget();
}

```

```

ablak1()
{
int bfs,bfo,jas,jao;
int m;
bfs=11;
bfo=44;
jas=17;
jao=70;
while(!kbstat()) /* addig, mig nem nyomunk billentyut */
{

if((fc2=windsave(bfs,bfo,jas,jao))==NULL)
{

```



```

        erase();
        printf("WINDSAVE ERROR (SELECT) !");
        fioclose(file);
        exit(0);
    }

    windfram(fc2,1,MSGWIND);
    cupo(bfs+1,bfo+2);
    aputtys("A most futó program a ",MSGWIND);
    cupo(bfs+2,bfo+9);
    aputtys("D E M O ",MSGWIND);
    cupo(bfs+3,bfo+2);
    aputtys("program, amely bemutatja",MSGWIND);
    cupo(bfs+4,bfo+2);
    aputtys("a rendszer lehetosegeit",MSGWIND);
    cupo(bfs+5,bfo+2);
    aputtys("az Ön közreműködésével.",MSGWIND);
    if((m=kbstat())!=0)
        break;
    delay(3000);
    windclose(fc2);
    if((m=kbstat())!=0)
        break;
    delay(1500);
}

curstype(savecurs)>>8,savecurs & 0xff);/* kurzorvisszaállítás */
}

filedef()
{
    int cs,i;          /* a file-struktúra segédpufferei */
    char poss[6],recl[3],orde[3],kudb[3];
    recl[0]='\0';
    orde[0]='\0';
    kudb[0]='\0';
    screen("A FAJLSTRUKTÚRA DEFINÍCIÓJA"); /* képernyő föcimmel */
    msgwrite("EXIT -> F1"); /* üzenet az alsó sorba */
    cupo(8,10);
    aputtys("A rekordok hossza =",cwritetext);
    while(fkl[0].length<1) /* amíg a file hossza < 1 */
    {
        cs=getfield(2,8,30,2,recl); /* mezőszerkesztés */
        if(cs==F1)
            {

```



```

        erase();
        fioclose(file);
        exit(0);
    }
    fkl[0].length=atoi(recl);
    if(fkl[0].length>60)
    {
        errorw("A DEMO-hoz maximum 60-as rekordhosszt kerek");
        fkl[0].length=0;
    }
}

cupo(10,10);
aputtys("Novekvo/csokk(1/0)=",cwritetext);    /* rendezettseg */
fkl[0].order=2;
while((fkl[0].order!=0)&&(fkl[0].order!=1))
{
    cs=getfield(2,10,30,1,orde);
    if(cs==F1)
    {
        erase();
        fioclose(file);
        exit(0);
    }
    fkl[0].order=atoi(orde);    /* stringbol integert csinal*/
}

cupo(12,10);
aputtys("Kulcsdarabok szama=",cwritetext);
while(fkl[0].keynum<1)    /* ahany kulcsdarab van*/
{
    cs=getfield(2,12,30,1,kudb);
    if(cs==F1)
    {
        erase();
        fioclose(file);
        exit(0);
    }
    fkl[0].keynum=atoi(kudb);
}
for(i=0;i<fkl[0].keynum;i++)
{
    poss[0]='\0';
    cupo(14,4);

```



```

    itoa(poss,i+1,2);
    aputtys(poss,cwritetext);
    aputtys(". Kulcsdarab",cwritetext);
    cupo(14,25);
    itoa(poss,fkl[0].keypieces[i].keypos,3);
    aputtys("Kezdopont= ",cwritetext);
        /* a kulcsdarab kezdopoz. a rekordban*/
    fkl[0].keypieces[i].keypos=-1;
    while(fkl[0].keypieces[i].keypos<0)
    {
        cs=getfield(2,14,35,2,poss);
        if(cs==F1)
        {
            erase();
            fioclose(file);
            exit(0);
        }
        fkl[0].keypieces[i].keypos=atoi(poss);
        if(fkl[0].keypieces[i].keypos>fkl[0].length)
        {
            errorw(
"Kulcsdarab kezdopontja nem lehet nagyobb a rekordhossznel");
            fkl[0].keypieces[i].keypos=-1;
        }
    }
    cupo(14,40);
    itoa(poss,fkl[0].keypieces[i].keylength,3);
    aputtys("Hossz= ",cwritetext);
    fkl[0].keypieces[i].keylength=0; /* a kulcsdarab hossza */
    while(fkl[0].keypieces[i].keylength<1)
    {
        cs=getfield(2,14,46,2,poss);
        if(cs==F1)
        {
            erase();
            fioclose(file);
            exit(0);
        }
        fkl[0].keypieces[i].keylength=atoi(poss);
        if(fkl[0].keypieces[i].keylength>fkl[0].length-
            fkl[0].keypieces[i].keypos)
        {

```



```

        errorw("A kulcsdarab a rekordon tul er");
        fkl[0].keypieces[i].keylength=0;
    }
}

cupo(16,20);
aputtys("Rekordszam= ",cwritetext);
rekszam=0;
while(rekszam<1) /* hany rekord legyen ? */
{
    cs=getfield(2,16,32,2,poss);
    if(cs==F1)
    {
        erase();
        fioclose(file);
        exit(0);
    }
    rekszam=atoi(poss);
}
msgdel();
}

fizolv() /* REREAD */
{
    int ve,cs;
    long i;
    char fizsor[4];
    screen("REREAD");
    msgwrite("EXIT -> F1  RUTINMENU -> F2");
    cupo(7,20);
    aputtys("Kerem a beolvasando rekord fizikai sorszamat !"
        ,cwritetext);
    fizsor[0]='\0';
    i=0;
    while(i==0)
    {
        cs=getfield(2,9,oszlopk,2,fizsor);
        if(cs==F1)
        {
            erase();
            fioclose(file);
            exit(0);
        }
    }
}

```



```

        if(cs==F2)
            return(0);
        i=atoi(fizsor);    /* i-be kerül a fizikai rekordszám */
    }
    fioclose(file);
    ve=reread(file,buff,i);
    fioopen(file);
    if(ve==0)
    {
        if(buff[0]=='!')
            errorw("Ez törölt rekord !");
        else
        {
            cupo(11,28);
            aputtys("A keresett rekord :",cwritetext);
            cupo(14,oszlopr);
            aputtys(buff,CMSGWIND);
            errorw("Nyomjon egy billentyut !");
        }
    }
}

```

```

fizir()                                /* REWRITE */
{
    int ve,cs;
    int i;
    long fizsorsz;
    char fizsor[4]; /* Fizikailag hanyadik rekordot irjuk felul */
    screen("REWRITE");
    msgwrite("EXIT -> F1   RUTINMENU -> F2");
    cupo(7,20);
    aputtys("Kerem a felulirando rekord fizikai sorszamat !"
            ,cwritetext);
    fizsorsz=0;
    while(fizsorsz==0)
    {
        cs=getfield(2,9,oszlopk,2,fizsor);
        if(cs==F1)
        {
            erase();
            fioclose(file);
        }
    }
}

```



```

        exit(0);
    }
    if(cs==F2)
        return(0);
    fizesorsz=atoi(fizesor);
}
fioclose(file);          /* a reread onalloan nyitja a file-t */
ve=reread(file,buff,fizesorsz);
if(buff[0]!='!')
{
    errorw("Ez torolt rekord, nem irhato felul !");
    fioopen(file);
    return(0);
}
if(ve==0)
{
    cupo(11,25);
    aputtys("Kerem az uj rekordot !",cwritetext);
    buff[0]='\0';
    while(strlen(buff)!=fkl[0].length)
    {
        cs=getfield(1,13,oszlopr,fkl[0].length,buff);
        if(cs==F1)
        {
            erase();
            exit(0);
        }
        if(cs==F2)
        {
            fioopen(file);
            return(0);
        }
        for(i=0;i<fkl[0].length;i++)
        {
            if(buff[i]>123)
                /* a rekord i-edik byte-ja>123*/
                errorw(
                    "Hibas adat, tul nagy ASCII kod!");
            buff[i]='\0';
            fioopen(file);
        }
    }
}

```



```

        return(0);
    }
}

    }
    ve=rewrite(file,buff,fizsorsz);
    if(ve==0)
        errorw("Felulirtam, nyomjon egy billentyut !");
}
fioopen(file);
}

beszur()                /* INSERT */
{
    int cs,j,ve,i;
    int karakter;
    screen("INSERT");
    msgwrite("EXIT -> F1  RUTINMENU -> F2");
    cupo(7,25);
    aputtys("Kerem a beszurando rekord kulcsat !",cwritetext);
    kulcs[0]='\0';
    while(strlen(kulcs)!=kulcshossz)
    {
        cs=getfield(1,9,oszlopk,kulcshossz,kulcs);
        if(cs==F1)
        {
            erase();
            fioclose(file);
            exit(0);
        }
        if(cs==F2)
            return(0);
        for(i=0;i<kulcshossz;i++)
        {
            if(kulcs[i]>123)
            {
                errorw("Hibas a kulcs, tul nagy ASCII kod!");
                kulcs[i]='\0';
                break;
            }
        }
    }
}
ve=search(file,kulcs,buff);

```



```

if(ve==0)
{
    for(j=0; j<fkl[0].length; j++)
    {
        karakter=rand()/762+47; /* a veletlenszamok */
        buff[j]=karakter;      /* transzformalasa */
    }
    kulcsbol(kulcs,buff); /* a rekordba masolja a kulcsot */
    cupo(11,28);
    aputtys("A kulcshoz generalt rekord :",cwritetext);
    cupo(14,oszlopr);
    aputtys(buff,CMSGWIND);
    insert(file,buff);
    errorw("Beszurtam, nyomjon egy billentyut !");
}

else
{
    cupo(11,28);
    aputtys("A megtalalt kulcsu rekord :",cwritetext);
    cupo(14,oszlopr);
    aputtys(buff,CMSGWIND);
    errorw("Van mar ilyen kulcsu rekord");
}

}

kulcsbol() /* segedrutin az insert-hez */
/* a rekordpufferbe masolja a megadott kulcsot */
/* az egyszeruseg kedveert csak a kulcsot
   kertuk be az insertbe ! */

{
    int j,m;
    m=0;
    for(j=0; j<fkl[0].keynum; j++) /* Kulcsdarabig masolunk */
    {
        strncpy(&buff[fkl[0].keypieces[j].keypos],&kulcs[m],
                fkl[0].keypieces[j].keylength);
        m+=fkl[0].keypieces[j].keylength;
    }
}

```



```

keres()          /* SEARCH */
{
    int i,cs,ve;
    char fizsor[4];
    screen("SEARCH");
    msgwrite("EXIT -> F1   RUTINMENU -> F2");
    cupo(7,25);
    aputtys("Kerem a keresendo rekord kulcsat !",cwritetext);
    kulcs[0]='\0';
    while(strlen(kulcs)!=kulcshossz)
    {
        cs=getfield(1,9,oszlopk,kulcshossz,kulcs);
        if(cs==F1)
        {
            erase();
            fioclose(file);
            exit(0);
        }
        if(cs==F2)
            return(0);
        for(i=0;i<kulcshossz;i++)
        {
            if(kulcs[i]>123)
            {
                errorw("Hibas a kulcs, tul nagy ASCII kod!");
                kulcs[0]='\0';
                break;
            }
        }
    }
    ve=search(file,kulcs,buff);
    if(ve<0)          /* Nem sikerult a search */
        return(0);
    if(ve==0)
    {
        if(buff[0]=='!')
        {
            errorw
            ("Nincs ilyen kulcsu rekord, ez a file vegere kerulne");
            return(0);
        }
        else
    }

```



```

        {
            errorw("Nincs ilyen kulcsu rekord");
            cupo(11,28);
            aputtys("A kovetkezo rekord kulcsa:",cwritetext);
            kulcsval(file,kulcs,buff);
            cupo(13,oszlopk);
            aputtys(kulcs,CMSGWIND);
            cupo(15,30);
            aputtys("A kovetkezo rekord :",cwritetext);
            cupo(17,oszlopr);
            aputtys(buff,CMSGWIND);
        }
    }

else
    {
        cupo(12,15);
        aputtys("A keresett rekord :",cwritetext);
        cupo(15,oszlopr);
        aputtys(buff,CMSGWIND);
        itoa(fizsor,ve,2);
        cupo(18,10);
        aputtys("Fizikai sorszama az adatfile-ban :",cwritetext);
        aputtys(fizsor,CMSGWIND);
    }

errorw("Nyomjon egy billentyut !");
}

gorget() /* NEXT */
/* a megadott hatarak koze eso rekordokat listazza ki */
{
    int i,ve,cs,m;
    char *malloc(),*kulcs2; /* a kulcs felso hatara */
    kulcs2=malloc(kulcshossz+1);

    screen("NEXT");
    msgwrite("EXIT -> F1   RUTINMENU -> F2");
    cupo(5,25);
    aputtys("Kerem a listazasi tartomanyt !",cwritetext);
    cupo(7,29);
    aputtys("Also hatar kulcsa :",cwritetext);
    kulcs[0]='\0';

```



```

while(strlen(kulcs)!=kulcshossz)
{
    cs=getfield(1,9,oszlopok,kulcshossz,kulcs);
    if(cs==F1)
    {
        erase();
        fioclose(file);
        exit(0);
    }
    if(cs==F2)
    {
        free(kulcs2);
        return(0);
    }
}
cupo(11,29);
aputtys("Felső hatar kulcsa :",cwritetext);
kulcs2[0]='\0';
while(strlen(kulcs2)!=kulcshossz)
{
    cs=getfield(1,13,oszlopok,kulcshossz,kulcs2);
    if(cs==F1)
    {
        erase();
        fioclose(file);
        exit(0);
    }
    if(cs==F2)
    {
        free(kulcs2);
        return(0);
    }
}
ve=strcmp(kulcs,kulcs2);          /* ket kulcs hasonlitasa */
if(ve>0&&fkl[0].order==1)        /* az 1. kulcs nagyobb es */
/* a rendezettseg novekvo */

{
    errorw("Hibas hatarok");
    free(kulcs2);
    return(0);
}

```



```

if(ve<0&&fkl[0].order==0)          /* a 2. kulcs nagyobb es */
                                     /* a rendezettseg csokkeno */
{
    errorw("Hibas hatarok");
    free(kulcs2);
    return(0);
}
ve=search(file,kulcs,buff);
if(ve==0&&buff[0]=='!')
{
    errorw("Nincs listazando tetel");
    free(kulcs2);
    return(0);
}
kulcsval(file,kulcs,buff);          /* mar az also hatar nem kell */
i=strcmp(kulcs,kulcs2);
if(i>0&&fkl[0].order==1)
{
    errorw("Nincs listazando tetel !");
    free(kulcs2);
    return(0);
}
if(i<0&&fkl[0].order==0)
{
    errorw("Nincs listazando tetel !");
    free(kulcs2);
    return(0);
}
cupo(15,28);
aputtys("A rekord :",cwritetext);
ve=next(file,buff);                /* az elso next ugyanazt a rekordot */
                                     /* hozza be, mint az elobbi search */
while(1)                            /* Van listazando tetel */
{
    cupo(17,oszlopr);
    aputtys(buff,CMSGWIND);
    msgwrite(
        "Kovetkezo rekord -> ENTER  EXIT -> F1  RUTINMENU ->F2");
    kbflush();
    while(1)
    {
        /* billentyulekerdezés */
        m=kbget();

```



```

        cs=m >> 8;
        if(cs==F1)
        {
            erase();
            fioclose(file);
            exit(0);
        }
        if(cs==F2)
        {
            free(kulcs2);
            return(0);
        }
        if(cs==ENTER)
            break;
    }
    if(ve==1)        /* a next 1-gyel ter vissza, */
    {                /* ha vege van az allomanynak */
        errorw("Nincs tobb !");
        free(kulcs2);
        return(0);
    }
    ve=next(file,buff); /* a kovetkezo rekord beolvasasa */
    kulcsval(file,kulcs,buff); /* a kulcs kivetele buff-bol */
    i=strcmp(kulcs,kulcs2); /* a kulcs hasonlitasa
                               a felso hatar kulcsaval */
    if(i>0&&fkl[0].order==1)
    {
        errorw("Nincs tobb !");
        free(kulcs2);
        return(0);
    }
    if(i<0&&fkl[0].order==0)
    {
        errorw("Nincs tobb !");
        free(kulcs2);
        return(0);
    }
}

```

```

torles()                /* DELETE */
{

```



```

int i,cs,ve;
screen("DELETE");
msgwrite("EXIT -> F1   RUTINMENU -> F2");
cupo(7,25);
aputtys("Kerem a torlendo rekord kulcsat !",cwritetext);
kulcs[0]='\0';
while(strlen(kulcs)!=kulcshossz)
{
    cs=getfield(1,9,oszlopk,kulcshossz,kulcs);
    if(cs==F1)
    {
        erase();
        fioclose(file);
        exit(0);
    }
    if(cs==F2)
        return(0);
    for(i=0;i<kulcshossz;i++)
    {
        if(kulcs[i]>123)
        {
            errorw(
                "Hibas a kulcs, tul nagy ASCII kod!");
            kulcs[0]='\0';
            break;
        }
    }
}
ve=search(file,kulcs,buff);
if(ve==0)
    errorw("Nincs ilyen kulcsu rekord !");
else
{
    cupo(12,15);
    aputtys("A torolni kivant rekord :",cwritetext);
    cupo(15,oszlopr);
    aputtys(buff,MSGWIND);
    delete(file,buff);
    errorw("Toroltem, nyomjon egy billentyut !");
}
}

```



```

msgwrite(sz)
char *sz;
{
    int i;
    i=40-strlen(sz)/2;
    cupo(23,1);
    awrchar(78,32,cwritetext);
    cupo(23,i);
    aputtys(sz,cwritetext);
}

```

```

msgdel()
{
    cupo(23,1);
    awrchar(78,32,7);
}

```

```

int screen(sz)
char *sz;
{
    char date[12];
    int i,j,len,n,kod,c,sor,max,osz;
    erase();
    aframe(0,0,24,79,0,cwritescr);
    aframe(2,1,2,78,0,cwritescr);
    aframe(22,1,22,78,0,cwritescr);
    cupo(2,1);
    awrchar(1,205,cwritescr);
    cupo(2,78);
    awrchar(1,205,cwritescr);
    cupo(22,1);
    awrchar(1,205,cwritescr);
    cupo(22,78);
    awrchar(1,205,cwritescr);
    cupo(1,67);
    getdat(date);
    aputtys(date,cwritescr);
    cupo(1,1);
    aputtys(ceg,cwritescr);
    len=strlen(sz);
    i=(80-len)/2;
}

```



```

cupo(1,i);
aputtys(sz,ctitlescr);
}

```

```

putfield(row,col,maxlength,str)
    /* maxlength hosszban kiirja az str -t row,col-ba */

```

```

int row;
int col;                /* kezdó sor-oszlop */
int maxlength;
char *str;
{
    cupo(row,col);
    awrchar(maxlength,32,cetext);
    aputtys(str,cetext);
}

```

```

getfield(type,row,col,maxlength,str)

```

```

/*      -   A row-col poziciótól egy legfeljebb maxlength hosszú
          stringet olvas str-be, f1-f2,f7,f8 -ig
          vagy return-ig.
      -   Ez a kód a scan, a visszaadott érték
      -   Az str-nek '\0'-t mindig tartalmaznia kell
      -   type = 0  numerikus mező
                = 1  alfanumerikus
                = 2  egész
                = 3  hó
                = 4  nap
                = 5  kód */

```

```

int row;                /* a kezdő sor          */
int col;                /* a kezdő oszlop        */
int maxlength;          /* a string max. hossza  */
int type;               /* a mező típusa         */
char *str;              /* a visszaadott string  */
{
    int i,l,d;
    int cchar, cscan;

```

```

    cupo(row,col);
    setattr(maxlength,cedit);

```



```

i=0;
cimke;;
while (i<maxlength)
{
    l=kbget();
    cscan = l>>8;
    cchar = l&255;
    if (cchar==0)
    {
        switch (cscan)
        {
            case LEFT:
            case F9:
                i--;
                if (i < 0)
                    i=0;
                cupo(row,col+i);
                goto tovabb;
            case RIGHT:
            case F10:
                i++;
                if (i > strlen(str))
                    i--;
                if (i>=maxlength)
                    i--;
                cupo(row,col+i);
                goto tovabb;
            case F1:
            case HOME:
            case F2:
            case END:

                if (type==5||type<3||
                    (type==3&&atoi(str)<13))
                    goto veg;
                if (type==4&&atoi(str)<32)
                    goto veg;
            default;;
                goto tovabb;
        }
    }
}
else /* a char kod nem 0 */
{

```



```

        if (cscan==ENTER)
        {
            if (type==5||(type<3||(type==3&&
            atoi(str)<13)||(type==4&&atoi(str)<32))
                goto veg;
        }
    else
    {
        if (cchar !=8)
        {
            /* numerikus */
            if (type==0&&(cchar<43||cchar>57
            ||cchar==47||cchar==44))
                goto tovabb;

            /* alfabetikus */
            if (type==1&&(cchar<32||
            cchar>126||cchar==36))
                goto tovabb;

            /* egesz,ho,nap */
            if (type>1&&(cchar<48||cchar>57))
                goto tovabb;
            if(str[i] =='\0')
                str[i+1] = str[i];
            str[i] = cchar;
            cupo(row,col+i);
            awrchar(1,cchar,credit);
            if(i<maxlength-1)
            {
                i++;
                cupo(row,col+i);
            }
        }
    }
    else
    {
        i--;
        if (i >= 0)
        {
            l=0;
            do
            {
                ++l;
            }
        }
    }
}

```



```

if(k<10)
    zdb=k;
jas=bs2+2+zdb;
if((fc=windsave(bs2,bo2,jas,jao))==NULL)
{
    erase();
    printf("WINDSAVE ERROR (SELECT) !");
    exit(0);
}
windfram(fc,1,CWINDFRAME);
cupo(jas,bo2+(jao-bo2+1)/2-5);
aputtys("Dn,Up,Enter",CWINDFRAME);
sound(1200);delay(800);nosound();
ikezd=0;
i=0;
cs=0;
while(cs!=ENTER)
{
    for(j=0;j<zdb+1;j++)
    {
        cupo(bs2+1+j,bo2+1);
        iakt=ikezd+j;
        if(iakt>k)
            iakt-=k+1;
        aputtys(s[iakt],CWINDTEXT);
    }
    cupo(bs2+1+i,bo2+1);
    iakt=ikezd+i;
    if(iakt>k)
        iakt-=k+1;
    aputtys(s[iakt],CWINDMARK);
    while(1)
    {
        kbflush();
        cs=kbget(>>8);
        if(cs==ENTER)
        {
            mut=ikezd+i;
            if(mut>k)
                mut-=k+1;
            break;
        }
    }
}

```



```

        if(cs==DOWN)
        {
            i++;
            if(i>zdb)
            {
                i=zdb;
                ikezd++;
                if(ikezd>k-zdb)
                    ikezd=k-zdb;
            }
            break;
        }
        if(cs==UP)
        {
            i--;
            if(i<0)
            {
                i=0;
                ikezd--;
                if(ikezd<0)
                    ikezd=0;
            }
            break;
        }
    }

    windclose(fc);
    curstype(savecurs >> 8 , savecurs & 0xFF);
    return(mut);
}

```

```

itoa(buf,iert,size)    /* iert egeszertek konverzioja buff-ba */
char *buf;
int iert,size;
{
    static char tabla[13]="0123456789 ";
    long cc;
    int cs,i,cd,mut,ejel,j;
    mut=0;
    j=0;

```



```

eje1=0;
if(iert<0)
{
    iert=-iert;
    eje1=1;
}
cs=iert;cc=1;
for (i=1;i<size;i++)
{
    buf[i]=' ';
    cc=cc*10;
}
for (i=0;i<size;i++)
{
    cd=cs/cc;
    if(mut==0&&cd==0)
        cd=10;
    else
    {
        mut=1;
        buf[j]=tabla[cd];
        j++;
    }
    if(cd==10)
        cd=0;
    cs=cs-cc*cd;
    cc=cc/10;
}
if(eje1==1)
{
    for(i=j;i>0;i--)
        buf[i]=buf[i-1];
    buf[0]='-';
}
if(iert==0)
    buf[0]='0';
buf[size]='\0';
return(0);
}

/* end DEMO.C */

```


8. A CFIO rendszer hibajelzései

8.1. Az eljárások hibaüzenetei

A táblázatban az egyes eljárások angol nyelvű hibaszövegei találhatóak a hibakóddal, illetve a szöveg magyar jelentésével együtt. Az táblázat az eljárások ábécé sorrendjében tartalmazza a hibaszövegeket. A kulcswal és a fioclose eljárásokat azért nem tartalmazza a táblázat, mert önálló hibajelzésük nincs.

Eljárás	Kód	Hibaszöveg	Jelentés
BUILD	1	Index file is update!	Az index karbantartott.
	0	Normal building!	Hibátlan build.
	-1	File not exist !	A file nem létezik.
	-2	File is empty!	A file üres.
	-3	File definition error or not fix recordlength!	File-definíciós hiba, v. hibás a rekordhossz.
	-4	Too large file! Max. record number 65024!	Legfeljebb 65024 rekord! lehet egy állományban.
	-5	Not enough memory for building file structure!	Nincs elég memória az indexfile építéséhez.
	-6	I couldn't open the new indexfile!	Nem lehet megnyitni az az új indexfile-t.
	-7	Not enough memory for buffers!	Nincs elég memória a pufferek számára.
	-8	Not enough memory for page buffer!	Nincs elég memória a lappuffer számára.

Eljárás	Kód	Hibaszöveg	Jelentés
DELETE	0	Successful delete!	Sikeres törlés.
	-1	Invalid page number!	A fiopage érték hibás.
	-2	Invalid record number!	A fionumber hibás.
	-3	Not enough memory for buffers!	Nincs elég memória a pufferek számára.
	-4	I couldn't make the new indexfile!	Nem lehet elkészíteni az új indexfile-t.
FCREAT	2	Index file exist!	Az indexfile létezik.
	1	Data file exist!	Az adatfile létezik.
	0	Successful file creat!	Sikeres létrehozás.
FIOOPEN	0	Successful open!	Sikeres nyitás.
	-1	The indexfile not exist!	Az indexállomány nem létezik.
	-2	The datafile not exist!	Az adatállomány nem létezik.
INSERT	0	Successful insert!	Sikeres beszúrás.
	-1	Too many records!	Túl sok rekord.
		Maximumnumber is 65024!	Legfeljebb 65024 lehet.
	-2	Invalid page number!	A fiopage érték hibás.
	-3	Invalid record number!	A fionumber hibás.

Eljárás	Kód	Hibaszöveg	Jelentés
NEXT	1	EOF on file!	Az állomány utolsó rekordját olvastuk be. Következik az első.
	0	Successful read!	Sikeres olvasás. A paraméterek a következőre fognak mutatni.
	-1	Invalid page number!	A fiopage érték hibás.
	-2	Invalid record number!	A fionumber hibás.
PACK	0	Successful packing!	Sikeres tömörítés.
	-1	File not exist!	A file nem létezik.
	-2	File is empty!	A file üres.
	-3	I couldn't open workfile on default!	Nem tudok munkaállományt nyitni a lemezen.
	-4	Not enough memory for buffers!	Nincs elég memória a pufferek számára.
	-5	I couldn't open the new indexfile!	Nem tudom megnyitni az új indexfile-t.
REREAD	0	Successful reading!	Sikeres olvasás.
	-1	I couldn't open file for reread!	Nem tudom az állományt olvasásra megnyitni.
	-2	Invalid record number!	Hibás rekordsorszám.
	-3	File position error!	File-pozicionálási hiba!

Eljárás	Kód	Hibaszöveg	Jelentés
REWRITE	0	Successful rewrite!	Sikeres felülírás.
	-1	I couldn't open file for rewrite!	Nem tudom az állományt felülírásra megnyitni.
	-2	Invalid record number!	Hibás rekordsorszám.
	-3	File position error!	File-pozicionálási hiba!
	-4	Invalid key! I can't rewrite!	Megváltozott a kulcs. Nem tudom felülírni.
SEARCH	+X	X-th record founded!	Az X.rekordban talált ilyen kulcsértéket.
	0	This key isn't in file!	Ez a kulcs még nem sze- repel az állományban.
	-1	File definition error or not fixed reclength!	File-definíciós hiba v. hibás a rekordhossz.

8.2. A utility programok hibajelzése.

Utility	Üzenet	Magyarázat
DUMP.EXE	Ez a file nem létezik!	Az elemezendő file nem létezik.
	Használja így: dump file-nev!	Nem adta meg argumentként az elemezendő indexfile nevét.
	Ez a file üres!	Üres állomány nem elemezhető.
	Ez a file nem SFIO indexfile!	A file nem elemezhető, mert a tartalma biztosan nem indexfile.
BUILD.EXE	Kész!	Az indexépítés sikeresen befejeződött.
	Épitem az új indexfile-t!	Megkezdődött az állomány újjáépítése.
PACK.EXE	Kész!	A tömörítés sikeresen befejeződött.
	Tömörítem az állományt!	Megkezdődött az állomány tömörítése.



AKADÉMIAI KIADÓ ÉS NYOMDA
Felelős kiadó: Hazai György főigazgató
Felelős szerkesztő: Votisky Zsuzsa
A borítót Szabadi Zoltán tervezte
ISBN 963 05 4741 4
A kézirat lezárva: 1987. szeptember
OPTIMUM – MGKSZ NY. 87. 358

CFIO -- mint C nyelvű File I/O,

vagyis C nyelven írt file-kezelő eljárások gyűjteménye

Az általános célú C programozási nyelv csak a legegyszerűbb állománykezelő funkciókat látja el. A felhasználói programok döntő többsége azonban adatfeldolgozás, hiszen minden ügyviteli program különféle nyilvántartó, rendező, kereső, stb. funkciót lát el.

A fiatal programozó testvérpár, Marjai György és Marjai Tamás egy olyan CFIO programkönyvtárat állítottak össze, amely a C nyelven programozók számára kényelmessé teszi az adatfeldolgozó rendszerek készítését.

A CFIO rendszer IBM/PC/XT/AT kompatibilis számítógépen a DOS 2.xx és DOS 3.xx operációs rendszerekben a Microsoft C 4.00 programnyelvi környezetben működik.

A programlemez a programkönyvtáron kívül az ismerkedést és betanulást segítő demonstrációs programot, valamint segédprogramokat tartalmaz.

A CFIO rendszer IBM kompatibilis hálózatokra való bővítése az SFIO rendszer.

A CFIO bemutató programja a CEX kiterjesztett C könyvtár segítségével készült.

A leírás a DOG dokumentáció generátorral készült.

Fogy. ár: 8000 Ft.

