

8

PRELUCRAREA ETICHETELOR DE FIȘIER ȘI A ALTOR INFORMAȚII EXTERNE MEMORATE ÎN DISCURI

Sistemul de operare gestionează o structură arborescentă de [sub]directoare și fișiere. La fiecare nivel din arborescență există tabele care încorporează informații de legătură și identificare a ascendenților și descendenților. În suportul extern, un fișier are, pe lângă partea de date, o intrare în [sub]directorul părinte, numită în continuare și *etichetă*. Aceasta memorează caracteristicile externe ale fișierului, cum ar fi: numele extern, extensia, atributele, data și ora ultimei scrieri în fișier etc. Programele utilizatorului pot avea acces la unele informații memorate în etichetele fișierelor și în alte tabele din structura DOS a discurilor, prin funcții și proceduri adecvate.

8.1 Structura fizică a discurilor DOS

Un disc magnetic, în structură DOS, este compus din următoarele părți: tabele de alocare a fișierelor, tabela directorului rădăcină, zona de date a fișierelor. În plus, discul sistem mai conține zona programului de încărcare a sistemului de operare (BOOT), localizată pe primul sector al primei piste. Fiecare fișier (subdirector) are o intrare (tip etichetă) în directorul (subdirectorul) părinte.

♦ **Tabela de alocare a fișierelor (FAT)** este împărțită în câmpuri care corespund *cluster*-elor de pe disc. *Cluster*-ul este format din unul sau mai multe sectoare și reprezintă unitatea de alocare pe disc. Numărul de sectoare ale unui cluster este memorat în blocul de parametri ai BIOS și diferă de la un tip de disc la altul (flexibil sau Winchester) și de la un tip de calculator la altul. Oricum, el este o putere a lui doi.

Exemplu:

8.1. Disc flexibil simplă față - 2^0 sectoare/cluster; disc flexibil dublă față - 2^1 sectoare/cluster; disc Winchester pentru PC/AT - 2^2 sectoare/cluster; disc Winchester pentru PC/XT - 2^3 sectoare/cluster.

Câmpurile din **FAT** au 16 biți (un cuvânt), prin care se pot adresa 2^{16} clustere (la unele sisteme, câmpurile din **FAT** au 12 biți).

Primul cuvânt din **FAT** (cu numărul zero) conține identificatorul tabelii de alocare, numit și descriptorul suportului. El indică, codificat, tipul de disc utilizat. Al doilea cuvânt din **FAT** (cu numărul unu) conține $FFFF_{16}$. Începând de la cuvântul cu numărul doi, sunt memorate lanțuri de alocare pentru fiecare fișier. Un cuvânt din **FAT** poate avea următorul conținut:

0000_{16} - cluster-ul corespunzător este disponibil;

$FFF0_{16}$ - $FFF6_{16}$ - cluster-ul corespunzător este rezervat;

$FFF7_{16}$ - cluster-ul corespunzător este defect;

$FFF8_{16}$ - $FFFF_{16}$ - sfârșitul lanțului de alocare;

0002_{16} - $FFEF_{16}$ - adresa următorului cuvânt din **FAT**, corespunzător următorului cluster ocupat de datele fișierului.

Lanțul de legături este realizat prin aceea că fiecare cuvânt indică numărul următorului cuvânt din lanț. Fișierul nu ocupă o zonă continuă în disc. El are cel puțin un cluster (când are un singur cluster, cuvântul corespunzător din **FAT** este $FFFF_{16}$).

Exemplu:

8.2. În figura 8.1 se prezintă o tabelă de alocare și lanțul de legături realizat pentru un fișier, **Fis**.

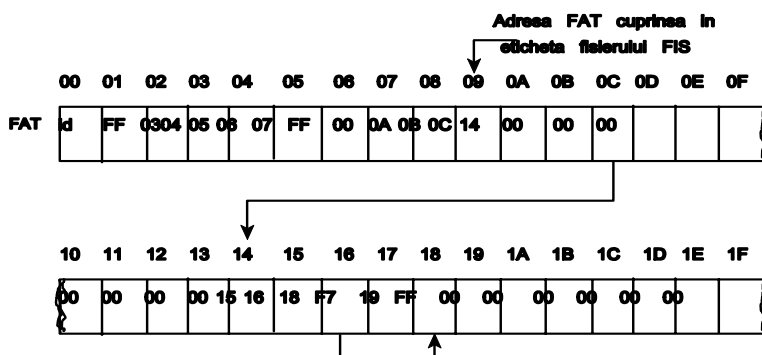


Fig. 8.1 Exemplu de tabelă de alocare

Pentru fișierul **Fis** sunt alocate cluster-ele **09-0C**, **14-16**, **18-19**. Cluster-ul **17** este defect. Cluster-ele **0D-13**, **1A-1F** sunt disponibile. Un alt lanț începe cu cluster-ul **02** și se termină cu cluster-ul **07**.

Prin algoritmi corespunzători, MS-DOS face conversia de la adresă de cluster la adresă de sector. Pentru a evita pierderea informațiilor, în cazul deteriorării unor câmpuri, în disc sunt păstrate două exemplare (adiacente) din **FAT**.

♦ **Zona tabelii directorului rădăcină (DIR)** urmează, în suport, tabelii de alocare a fișierelor și conține câte o intrare pentru fiecare fișier sau subdirector descendent. Intrarea este o zonă de 32 octeți cu o structură prestabilită. Fiecare disc are

un singur director rădăcină, căruia îi este asociată zona **DIR**. Dimensiunea și poziția tabelii **DIR** sunt fixe, prestabilite prin comanda **FORMAT**, în momentul formatării discului. Numărul de intrări în **DIR** și poziția tabelii în disc sunt memorate în blocul de parametri ai BIOS-ului. Dacă discul este de sistem, primele două intrări din **DIR** se referă la fișiere care conțin sistemul de operare (aceste fișiere sunt încărcate în memoria principală de către încărcător, care se află în sectorul zero al discului).

♦ **Zona de date a fișierelor (FILE)** conține partea de date a acestora și subdirectoarele. Subdirectoarele sunt tratate ca fișiere, cu articole speciale, de câte 32 octeți. Structura intrărilor în subdirectoare este identică cu cea a intrărilor în director. Numărul intrărilor în subdirectoare este nelimitat (spre deosebire de intrările în director). Sectoarele din zona de date sunt grupate în clustere care au câte un cuvânt corespunzător în **FAT**. Când un fișier (subdirector) este extins, se caută în **FAT** clustere libere (cuvinte cu conținutul 0000_{16}) și se prelungește lanțul de alocare. Dacă discul este de sistem, fișierele **IO.SYS** și **MSDOS.SYS** sunt primele în zona de date.

8.2 Structura intrărilor în [sub]directoare

Fiecărui fișier sau subdirector îi corespunde o intrare (etichetă) în [sub]directorul părinte. Intrările au lungimea 32 octeți. Subdirectoarele se comportă ca niște fișiere cu articole speciale, gestionate de DOS. Structura unei intrări este prezentată în tabelul 8.1.

♦ **Numele fișierului** este format din 8 caractere, primul având următoarele semnificații:

- **00** - Intrarea în director nu a fost folosită;
- **2E** - Intrarea corespunde unui [sub]director. $2E_{16}$ reprezintă caracterul ASCII ".". Zona 1A-1B a intrării conține mărimea subdirectorului, exprimată în numere de clustere. Dacă și octetul 01 al intrării este tot $2E_{16}$, zona 1A-1B conține adresa de început în **FAT** a lanțului de alocare a subdirectorului părinte (zona 1A-1B conține 0000_{16} dacă părintele este directorul rădăcină). În ambele situații, octeții 01-0A, respectiv 02-0A conțin caracterul spațiu;
- **E5** - Fișierul a fost șters;
- **Altă valoare** - Primul caracter al numelui de fișier (subdirector).

Din analiza structurii numelui se poate deduce că se selectează următoarele tipuri de intrări: intrare normală de fișier (subdirector) fiu; intrare de tip "."; intrare de tip "..". O mai bună precizare a tipului de intrare se realizează prin câmpul de atribute.

Tabelul 8.1 Structura intrărilor în [sub]director

Octeți	Semnificație
00 - 07	Numele fișierului
08 - 0A	Extensia fișierului
0B	Atributele fișierului
0C - 15	Octeți rezervați
16 - 17	Ora ultimei scrieri în fișier
18 - 19	Data ultimei scrieri în fișier
1A - 1B	Adresa de început în FAT a lanțului de alocare a fișierului
1C - 1F	Dimensiunea fișierului, în număr de octeți

♦ **Atributele fișierului** au semnificația din tabelul 8.2. Prin setarea sau ștergerea corespunzătoare a biților octetului **0B** pot fi obținute și alte combinații de atribute (cea mai utilizată este valoarea 03_{16} , care ar corespunde unui fișier *readonly* ascuns). Însurarea tuturor valorilor ($3F_{16}$) desemnează un fișier oarecare (orice fișier).

♦ **Ora ultimei scrieri în fișier** este memorată pe doi octeți și are structura din figura 8.2, unde:

O reprezintă ora, cu valori binare cuprinse în intervalul [0,23];

M reprezintă minutul, cu valori binare cuprinse în intervalul [0,59];

S reprezintă numărul de incremente de două secunde, exprimat în binar.

♦ **Data ultimei scrieri în fișier** este memorată pe doi octeți și are structura din figura 8.3, unde:

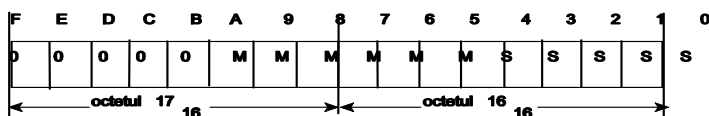


Fig. 8.2 Structura orei ultimei scrieri în fișier

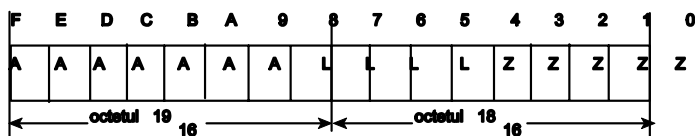


Fig. 8.3 Structura datei ultimei scrieri în fișier

- **A** este anul relativ la 1980, cu o valoare binară din intervalul [0-119];
- **L** este luna exprimată în binar (valori din intervalul [1,12]);
- **Z** este ziua exprimată în binar (valori din intervalul [1,31]).

Atât pentru ora, cât și pentru data ultimei scrieri există proceduri PASCAL care "împachetează", respectiv "despachetează", informația.

Tabelul 8.2 Atributele unui fișier (octetul **0B** al intrării în [sub]director)

Valoarea octetului 0B		S e m n i f i c a ț i e
Biți	Hexa	
76543210		
00000000	00	Fișier obișnuit. Asupra lui se pot realiza operații de citire și scriere.
00000001	01	Fișier <i>readonly</i> . Fișierul este protejat la scriere. Din el se poate doar citi. Fișierul nu poate fi șters.
00000010	02	Fișier ascuns (<i>hidden</i>). Fișierul nu poate fi găsit prin operații obișnuite de căutare în directori. Nu poate fi afișat prin comanda DIR din DOS și nu este disponibil prin comenzile COPY, DEL, REN, XCOPY.
00000100	04	Fișier de sistem (<i>system</i>). Se referă la sistemul de operare și este un fișier ascuns.
00001000	08	Identificator de volum. Un singur fișier, aflat în directorul rădăcină, are acest atribut.
00010000	10	[Sub]director (intrarea se referă nu la un fișier de date, ci la un [sub]director).
00100000	20	Fișierul este de tip arhivă (<i>archive</i>). Bitul este poziționat pe valoare 1 ori de câte ori are loc o scriere în fișier. Bitul redevine zero în urma unei procesări cu comenzile BACKUP, RESTORE, XCOPY din DOS.

♦ **Adresa de început în FAT** indică adresa primului cuvânt din tabela de alocare aferentă fișierului. Această adresă este mai mare ca unu. Drumul logic, de principiu, parcurs până la accesul la datele unui fișier cu identificatorul C:\D1\FIS este prezentat în figura 8.4.

8.3 Tratarea informațiilor din etichetele fișierelor

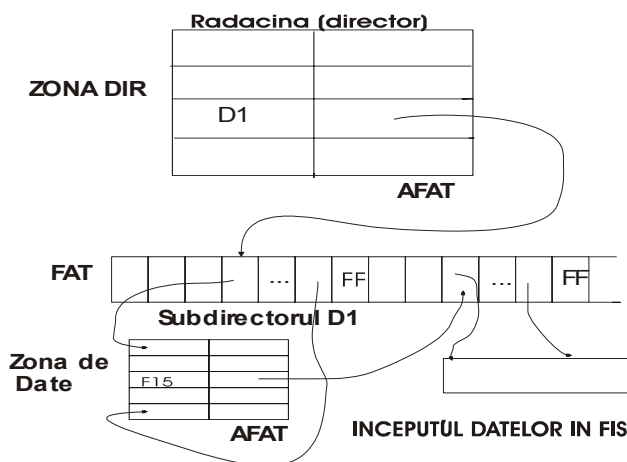


Fig. 8.4 Accesul la datele fișierului C:\D1\FIS

Eticheta de fișier (intrarea în [sub]director) conține câmpuri referitoare la nume, extensie, attribute, data și ora ultimei scrieri în fișier, adresa de început în **FAT**, dimensiunea fișierului. Etichetele sunt create la deschiderea cu **Rewrite** a fișierelor. De regulă, valorile câmpurilor nume, extensie, attribute și adresa de început în **FAT** sunt înscrise la crearea fișierului. Data și ora ultimei scrieri se modifică la fiecare nouă scriere în fișier. Dimensiunea fișierului se modifică la scrierea unui nou articol peste sfârșitul inițial de fișier (pentru fișierele **TEXT** la scrierea după o deschidere **Append**, iar pentru fișierele binare la scrierea unui articol cu pointer mai mare ca **FileSize(f)** inițial). Unele câmpuri pot fi modificate explicit, prin funcții și proceduri speciale. Astfel, **SetFAttr** poziționează attributele, **SetFTime** poziționează data și ora, **Rename** modifică numele [și extensia], **Truncate** modifică lungimea. Eticheta de fișier poate fi ștearsă cu procedura **Erase** (se șterge intrarea în [sub]director).

8.3.1 Prelucrarea atributelor unui fișier

În unit-ul **Dos** sunt definite proceduri pentru poziționarea sau citirea atributelor unui fișier. Corespunzător valorilor pe care le poate lua octetul **0B** din intrarea unui fișier în [sub]director, în unit-ul **Dos** sunt definite următoarele constante:

ReadOnly	= \$01	➤ fișier numai pentru citire;
Hidden	= \$02	➤ fișier ascuns;
SysFile	= \$04	➤ fișier de sistem;
Volumid	= \$08	➤ identificator de volum;
Directory	= \$10	➤ [sub]director;
Archive	= \$20	➤ fișier arhivă;
AnyFile	= \$3F	➤ orice fișier

Fișierelor de date li se pot asocia atributele **\$01**, **\$02**, **\$20**. În Turbo Pascal, un fișier deschis are, în mod implicit, atributul **\$20** (arhivă) deoarece în el s-a scris sau urmează să se scrie un articol. Fișierele binare cu atributul **\$01** pot fi prelucrate numai dacă sunt deschise cu **Reset** și dacă variabilei **FileMode** din unit-ul **System** i se atribuie, în program, valoarea zero. Fișierele **TEXT** cu atributul **\$01** pot fi doar citite (suportă numai deschiderea **Reset**).

Setarea pe o anumită valoare înseamnă, uneori, cumularea de atribute. De exemplu, dacă unui fișier cu atributul **\$10** (subdirector) i se setează valoarea 2 (hidden), se obține un subdirector ascuns (atributul **\$18**).

♦ **Procedura GetFAttr** returnează atributele unui fișier. Ea este definită astfel:
GetFAttr(VAR f; VAR attr:WORD)

F este variabila care indică fișierul, iar **attr** este variabila în care procedura depune valoarea atributelor.

Exemplu:

8.3. În secvența care urmează se citesc atributele fișierului extern 'TEST.DAT' și se afișează valorile acestora:

```
USES Dos;
VAR
  f:TEXT;
  i:BYTE;
BEGIN
  .....
  Assign(f, 'TEST.DAT');
  GetFAttr(f, i);
  Writeln(i);
  Reset(f);
  .....

```

♦ **Procedura SetFAttr** poziționează (setează) atributele unui fișier. Ea este

definită astfel:

SetFAttr(VAR f; VAR attr:WORD)

F este variabila care indică fișierul, iar argumentul real corespunzător lui **attr** este o expresie a cărei valoare reprezintă atributele și care va fi înscrisă extern în intrarea atașată fișierului în [sub]director.

Exemple:

8.4. În secvența care urmează, fișierului TEST1.DAT i se asociază atributul Readonly, iar fișierului TEST2.DAT atributul Hidden:

```
USES Dos;
VAR
  f1:TEXT;
  f2:FILE OF READ;
BEGIN
  .....
  Assign(f1,'TEST1.DAT');
  SetFAttr(f1,1); Reset(f1);
  .....
  Assign(f2,'TEST2.DAT');
  SetFAttr(f2,Hidden); Reset(f2);
  .....
```

8.5. În programul care urmează, un subdirector capătă atributul *ascuns*. După setare, valoarea atributelor subdirectorului este 18 (Directory+hidden).

```
USES Dos;
VAR
  f:FILE;
  nume_dir:STRING[79];
BEGIN
  Write('Director [n:\cale\director] : ');
  Readln(nume_dir);
  Assign(f,nume_dir);
  SetFAttr(f,2);
END.
```

Procedurile **GetFAttr** și **SetFAttr** au funcții echivalente cu comanda **ATTRIB** din **DOS**.

8.3.2 Prelucrarea datei și orei din eticheta unui fișier

În unit-ul **Dos** sunt definite proceduri pentru poziționarea sau citirea datei și orei ultimei scrieri în fișier. Procedurile prelucrează câmpurile **\$16-\$17** și **\$18-\$19** din eticheta fișierului (intrarea lui în [sub]director), ca o singură valoare de tip **LONGINT**. "Împachetarea", respectiv "despachetarea" în/din **LONGINT** se pot realiza cu procedurile **PackTime**, respectiv **UnpackTime**, definite în unit-ul **Dos**. Aceste proceduri utilizează tipul de date **DateTime** predefinit în unit-ul **Dos**, astfel:


```
DateTime = RECORD
    Year : WORD;
    Month : WORD;
    Day : WORD;
    Hour : WORD;
    Min : WORD;
    Sec : WORD;
END;
```

♦ **Procedura PackTime** "împachetează" un articol de tip DateTime într-o dată de tip LONGINT. Ea este definită astfel:

PackTime(VAR Dc:DateTime; VAR l:LONGINT)

Dc este data și ora, exprimate pe componente, care urmează să fie "împachetate" sub formă LONGINT în zona **DI**.

♦ **Procedura UnpackTime** "despachetează" o valoare de tip DateTime. Ea este definită astfel:

UnpackTime(DI:LONGINT, VAR Dc:DateTime)

DI este expresie de tip LONGINT care va fi "despachetată" pe componentele datei și orei în zona **Dc**.

♦ **Procedura GetFTime** returnează data și ora ultimei scrieri într-un fișier. Ea este definită astfel:

GetFTime(VAR f; VAR DI:LONGINT)

F este variabila care indică fișierul, iar **DI** este variabila în care se recepționează data și ora, sub formă "împachetată". Despachetarea lui **DI** se face cu procedura UnpackTime.

Exemplu:

8.6.

```
USES Dos;
VAR
    f1: FILE OF REAL
    z: DateTime;
    data_ora: LONGINT;
BEGIN
    .....
    Assign(f1, 'A:TEST.DAT');
    GetFTime(f1, Data_ora);
    UnpackTime(data_ora, z);
    IF Z.Year < 1994
    THEN Writeln('>> Varianta veche de fisier')
    ELSE Writeln('>> Varianta OK');
```

♦ **Procedura SetFTime** înscrie data și ora în eticheta fișierului. Ea este definită astfel:

SetFTime(VAR f; DI:LONGINT)

F este variabila care indică fișierul, iar **DI** este variabila care conține data și ora

sub formă împachetată, care se înscriu în intrarea fișierului în [sub]director. Forma împachetată poate fi obținută cu procedura **PackTime**.

Exemplu:

8.7.

```
USES Dos;
VAR
  f1:FILE OF REAL;
  z:DateTime; data_ora:LONGINT;
BEGIN
  .....
  ASSIGN (f1, 'A:TEST.DAT');
  z.Year:=1993; z.Month:=10; z.Day:=28;
  z.Hour:=11; z.Min:=20; z.Sec:=0;
  PackTime(z,Data_ora); SetFTime(f1,Data_ora);
  .....
```

8.3.3 Ștergerea fișierelor

În unit-ul **System** este definită procedura **Erase** care realizează ștergerea unui fișier existent (șterge eticheta din director). Declarația ei este:

Erase(VAR f)

Dacă fișierul extern, asignat lui **f**, nu există, execuția procedurii generează eroare de I/E (**IOResult** are valoare diferită de zero). Procedura **Erase** nu se poate executa asupra unui fișier deschis.

Exemplu:

8.8. Se citesc 100 de numere **x**. Elementele pozitive sunt memorate în fișierul cu tip **VECTOR.DAT**. Dacă toate numerele sunt negative, fișierul **VECTOR.DAT** nu trebuie să existe.

```
VAR
  Fis:FILE OF REAL;
  x:REAL; i,j:1..100; s:BOOLEAN;
BEGIN
  Assign(Fis, 'VECTOR.DAT'); Rewrite(Fis);
  j:=FALSE;
  FOR i:=1 TO 100 DO
    BEGIN
      Readln(x)
      IF x[i] >= 0 THEN
        BEGIN
          Write(Fis,x);
          j:=TRUE;
        END;
      END;
    Close(Fis);
    IF NOT j THEN Erase(Fis);
  END.
```

8.3.4 Redenumirea fișierelor

Redenumirea unui fișier existent (modificarea numelui fișierului din etichetă) se realizează cu procedura **Rename**, declarată în unit-ul **System**, astfel:

Rename(VAR f; nume_nou:STRING)

Numele intern, **f**, trebuie asignat anterior unui fișier (vechi) prin procedura **Assign**. Procedura **Rename** va redenumi acest fișier extern cu **nume_nou** (care este o expresie de tip **STRING**, cu lungimea maximă de 80 caractere). Procedura nu copiază fișiere, ci doar le redenumeste. De aceea, unitatea și calea fișierului vechi nu pot fi modificate. Procedura nu se aplică fișierelor deschise. Dacă fișierul extern, asociat lui **f** prin **Assign** (fișierul vechi), nu există, execuția procedurii generează eroare de I/E (**IOResult** are valoare diferită de zero). Dacă pe unitatea și în [sub]directorul unde este memorat fișierul vechi există deja un fișier cu numele **nume_nou**, procedura **Rename** produce eroare de I/E.

8.3.5 Trunchierea fișierelor

În unit-ul **System** este definită procedura **Truncate**, care trunchiază un fișier binar de la poziția curentă a pointerului. Ea este definită astfel:

Truncate(VAR f)

Procedura șterge din fișier articolele începând din poziția curentă a pointerului, până la sfârșit, modificând lungimea fișierului din etichetă. După execuția ei, funcția **Eof(f)** va returna valoarea **TRUE**.

8.4 Crearea și manipularea [sub]directoarelor

În unit-ul **System** există o serie de proceduri pentru crearea și ștergerea [sub]directoarelor, pentru definirea [sub]directorului curent și pentru obținerea valorii căii curente. Toate aceste proceduri au echivalent în comenzi DOS și funcționează identic cu acestea. În plus, având în vedere că subdirectoarele sunt tratate ca fișiere cu articole speciale, asupra lor pot fi aplicate funcțiile referitoare la citirea și setarea atributelor, datei și orei creării subdirectorului.

♦ **Procedura GetDir** returnează calea completă a subdirectorului curent (*path-ul*), dintr-o unitate de disc precizată. Declarația ei este:

GetDir(Drive:BYTE; VAR s:STRING)

Drive specifică unitatea de disc asupra căruia se aplică procedura și poate avea valorile: **0** - unitatea de disc curentă; **1** - unitatea de disc A; **2** - unitatea de disc B; **3** - unitatea de disc C etc.

S este variabila în care se recepționează calea completă a subdirectorului curent, începând cu rădăcina. Forma căii returnate este **n:cale**, unde **n** este numele unității de disc. Procedura este echivalentă cu comanda **CHDIR** (fără parametri) din **DOS**.

Exemple:

8.9. Dacă unitatea **C:** ar avea structura de directori din figura 1.4 și dacă directorul curent este D4, atunci procedura GetDir(3,Cale) ar returna, în variabila **Cale**, șirul: C:\D2\D4;

8.10. Procedura GetDir (9,Cale) returnează, în variabila **Cale**, șirul I:\, chiar dacă unitatea I:\ nu a fost definită pe discul fix.

♦ **Procedura ChDir** schimbă directorul curent. Ea are următoarea declarație:

ChDir(s:STRING)

S conține un șir de forma **[n:]cale** care definește noul subdirector curent. Calea poate fi precizată complet, pornindu-se de la rădăcină sau într-un mod relativ, folosindu-se formele construite cu următoarele caractere: \ (directorul rădăcină); . (directorul curent); .. (directorul părinte al celui curent).

Exemplu :

8.11. Dacă în structura de directoare din figura 1.4, directorul curent este D4 și se dorește ca D3 să devină curent, se poate proceda astfel:

ChDir('C:\D3'); sau

ChDir('\D3'); sau

ChDir('..'); ChDir('..'); ChDir('D3');

Când calea este specificată greșit se generează eroare (path not found). Procedura are funcție echivalentă cu comanda **CHDIR** din **DOS**.

♦ **Procedura Mkdir** creează un director în disc. Ea are următoarea declarație:

Mkdir(s:STRING)

S conține un șir de forma **[n:]cale** care definește subdirectorul care se creează. Lungimea maximă a căii, de la rădăcină până la nivelul dorit, nu trebuie să depășească 63 caractere, inclusiv caracterele \. Calea poate fi precizată complet, pornindu-se de la rădăcină sau relativ, folosindu-se formele prescurtate construite cu caracterele \, ., ..(vezi ChDir). Procedura are funcție echivalentă cu comanda **MKDIR** din **DOS**.

Exemplu:

8.12. Dacă în structura de director din figura 1.4 se dorește crearea unui subdirector D6, subordonat lui D3, se poate proceda astfel:

Mkdir('C:\D3\D6') ➤ de oriunde;

Mkdir('\D3\D6') ➤ de oriunde din unitatea C;

Mkdir('D6') ➤ din D3;

◆ **Procedura RmDir** șterge un subdirector gol. Ea are următoarea definiție:

RmDir(s:STRING)

S conține un șir de forma **[n:]cale** ce desemnează subdirectorul care se șterge. Calea poate fi specificată complet, pornindu-se de la rădăcină sau relativ, folosindu-se formele prescurtate construite cu caracterele \, ., .. (vezi ChDir).

Subdirectorul care se șterge nu trebuie să conțină fișiere sau subdirectoare atașate. În cazul în care subdirectorul nu există sau nu este gol se produce eroare de I/E. Directorul rădăcină și subdirectorul curent nu pot fi șterse. Procedura are funcție echivalentă cu comanda **RMDIR** din **DOS**.

Exemplu:

8.13. Se propune o procedură care asigură realizarea următoarelor operații, în funcție de valoarea unui parametru (**op**): determinarea directorului curent, schimbarea directorului curent, crearea unui director. Procedura se construiește în cadrul unui *unit* (**GestDir**), care poate fi extins și cu alte proceduri de interes general în lucrul cu directoare și/sau fișiere.

```
UNIT GestDir;
INTERFACE
PROCEDURE OpDir(op:CHAR; VAR d:STRING; VAR rez:INTEGER);
IMPLEMENTATION
PROCEDURE OpDir;
BEGIN
  CASE UpCase(op) OF
    'D': GetDir(0,d);
    'S': BEGIN
          {$I-} ChDir(d); {$I+}
          rez:=IOResult;
        END;
    'C': BEGIN
          {$I-} Mkdir(d); {$I+}
          rez:=IOResult;
        END;
  END;
END;
END;
END.
```

În continuare se prezintă un apelator al subprogramului, prin care se urmărește schimbarea directorului curent, cu salvarea directorului în uz de la momentul lansării, respectiv restaurarea lui la încheierea execuției programului. Dacă noul director nu există, se asigură posibilitatea creării lui, urmată de transformarea în director curent, în aceleași condiții. Specificatorul noului director este furnizat ca parametru în linia de comandă cu care este lansat în execuție programul. Numărul parametrilor de pe linia de comandă este determinat prin funcția **ParamCount:WORD**. Accesul la un anumit parametru se face prin funcția **ParamStr(n:WORD):STRING**, unde **n** este numărul de ordine al parametrului în lista din linia de comandă.

```
PROGRAM ExDir;
USES GestDir;
```

```
VAR  r : INTEGER;  c : CHAR;   dinc,dnou: STRING;
PROCEDURE DetDir;
VAR  dir: STRING;
BEGIN
  OpDir('d',dir,r);
  WriteLn('Director curent: ',dir);
END;
BEGIN
  WriteLn('Program in executie curenta: ',ParamStr(0));
  DetDir;
  IF ParamCount <> 0 THEN
    BEGIN
      dnou:=ParamStr(1);
      OpDir('d',dinc,r);
      OpDir('s',dnou,r);
      IF r <> 0 THEN
        BEGIN
          WriteLn('Director inexistent: ',ParamStr(1));
          Write('Se creeaza ca director nou?(Y/N): ');
          ReadLn(c);
          IF UpCase(c) = 'Y' THEN
            BEGIN
              OpDir('c',dnou,r);
              IF r = 0
                THEN OpDir('s',dnou,r)
                ELSE
                  BEGIN
                    WriteLn('>> Eroare la creare director.
Executie intrerupta!' );
                    RunError(r);
                  END;
            END;
          END;
        END;
      END;
      DetDir;
    END;
  {restul programului}
  IF ParamCount <> 0 THEN OpDir('s',dinc,r);
  DetDir;
END.
```

8.5 Căutarea fișierelor în [sub]directoare

În unit-ul **Dos** sunt definite proceduri care caută un fișier într-un [sub]director sau într-o listă de [sub]directoare. Procedurile utilizează tipurile de data **PathStr** și **SearchRec**, definite în unit-ul **Dos**, astfel:

```
PathStr=STRING[79];
SearchRec = RECORD
  Fill : ARRAY[1..21] OF BYTE;
  Attr : BYTE;
  Time: LONGINT;
  Size: LONGINT;
```

Name : STRING[12];
 END;

Semnificația câmpurilor tipului **SearchRec** este următoarea:

Fill: rezervat pentru sistemul de operare. Câmpul nu trebuie modificat în program.

Attr: atributele fișierului (valoarea octetului **\$0B** din intrarea fișierului în [sub]director).

Time: data și ora ultimei scrieri în fișier, sub formă împachetată (valoarea, ca LONGINT, a octeților **\$16-\$19** din intrarea fișierului în [sub]director).

Size: lungimea, în octeți, a fișierului (valoarea câmpului **\$1C-\$20** din intrarea fișierului în [sub]director).

Name: numele fișierului urmat de punct și extensia lui (valoarea câmpurilor **\$0-\$7** și **\$8-\$A** din intrarea fișierului în [sub]director).

♦ **Procedura FindFirst** caută într-un [sub]director prima intrare a unui fișier care are specificatorul și atributul precizate în lista de parametri. Ea are declararea:

FindFirst(F_extern:PathStr; Attr:Word; VAR zona:SearchRec)

F_extern este specificatorul extern al fișierului care este căutat. Specificatorul este format din cale, nume fișier și, eventual, extensie. Când calea lipsește se presupune [sub]directorul curent. Numele și extensia fișierului pot fi globale (formate cu caracterele * sau ?). **Attr** reprezintă valoarea atributelor fișierului care se caută. **Zona** este o variabilă de tipul **SearchRec**, definit în unit-ul **Dos**, care va conține informații despre fișierul **f_extern**, în cazul în care a fost găsit. Dacă **f_extern** nu este găsit, variabila **zona** rămâne nemodificată. Când **f_extern** este găsit, variabila **DosError** (definită în unit-ul **Dos**) va avea valoarea zero. În caz contrar, **DosError** are valoare diferită de zero (vezi §8.3).

♦ **Procedura FindNext** caută într-un [sub]director următoarea intrare a unui fișier care are specificatorul și atributul precizate la apelul anterior al procedurii **FindFirst**. De aici rezultă faptul că procedura **FindNext** poate fi utilizată numai dacă, anterior, a fost apelată procedura **FindFirst**.

Procedura este definită astfel:

FindNext(VAR zona:SearchRec)

Zona are aceeași semnificație ca la procedura **FindFirst**. Dacă următoarea intrare este găsită, variabila **DosError** (definită în unit-ul **Dos**) va avea valoarea zero. În caz contrar, **DosError** are valoare diferită de zero.

Exemplu:

8.14. Următorul program afișează toate fișierele cu extensia .DAT dintr-o cale precizată de la tastatură. Pentru fiecare fișier se afișează: numele, extensia, atributul, lungimea, data și ora ultimei scrieri în fișier.

```
PROGRAM EX14;  
USES Dos;  
VAR  
    Cale:String[79];  
    Zona:SearchRec;     {Tip definit in Dos}
```

```
Dc:DateTime;          {Tip definit in Dos}
BEGIN
  Write('Calea de cautat ([u:]\d1\d2\):');
  Readln(Cale); {Se citește calea de la tastatura; pentru
directorul curent se poate tasta ENTER}
  FindFirst(Cale+'*.DAT',$3F,Zona); {Cautarea primului fisier
cu extensia .DAT}
  IF DosError <> 0
    THEN Writeln('>> Nu exista fisier de tip .DAT')
    ELSE
      REPEAT
        UnpackTime(Zona.Time,Dc);
        Writeln(Zona.Name:12,' ',Zona.Attr:2,' ',Zona.Size:10,' ',Dc.Year
:4,' ',Dc.Month:2,' ',Dc.Day:2,' ',Dc.Hour:2,' H',Dc.Min:2,' M',Dc.
Sec:2,' S');
        FindNext(Zona);
      Until DosError=18;
END.
```

♦ **Funcția FSearch** caută un fișier într-o listă de [sub]directori. Ea este asemănătoare comenzii **PATH** din **DOS**. Funcția este definită astfel:

FUNCTION FSearch(Nume:PathStr; Lista:STRING):PathStr

Nume este variabilă de tip **PathStr** (definită în unit-ul **Dos**), care conține numele fișierului de căutat. **Lista** conține lista directoarelor în care se continuă căutarea, dacă fișierul nu a fost găsit în directorul curent. Căile specificate în listă trebuie separate prin caracterul ;.

Funcția returnează specificatorul extern al fișierului, în cazul în care îl găsește. Când fișierul este în directorul curent, specificatorul extern furnizat este format din nume și, eventual, extensie, iar, când fișierul este în alt director, specificatorul este complet (cale + nume [+ extensie]).

Când fișierul nu este găsit, funcția returnează șirul vid. Funcția nu verifică existența căilor. Când căile nu există, se returnează tot șirul vid.

Exemplu :

8.15. Programul care urmează caută un fișier într-o listă de directori. Când fișierul este găsit se afișează specificatorul extern. În caz contrar, se afișează mesajul >>Fișier negăsit.

```
PROGRAM EX15;
USES Dos;
VAR
  Nume:PathStr;
  x:STRING;
BEGIN
  Write('Numele extern al fisierului cautat (xxxxxxx.eee):');
  Readln(Nume);
  x:=FSearch(Nume,'C:\wp51\wp51\rosca\;tp\');
  IF x[0] = #0
    THEN Writeln('>>Fisier negasit')
    ELSE Writeln('>>Fisierul are specificatorul extern : ',x)
END.
```


8.6 Tratarea șirului referitor la specificatorul extern de fișier

În unit-ul **Dos** sunt definite o funcție și o procedură care prelucrează șirul de caractere referitor la specificatorul extern al unui fișier. Funcția (**FExpand**) și procedura (**FSplit**) nu fac nici o verificare sau căutare în suportul extern. Ele fac parte, mai degrabă, din categoria prelucrării unor șiruri de caractere, decât din cea de prelucrare a fișierelor.

În definirea funcției și procedurii se folosesc următoarele tipuri de date declarate în unit-ul **Dos**:

PathStr	= STRING[79];	➤ Specificator complet;
DirStr	= STRING[67];	➤ Unitate și cale;
NameStr	= STRING[8];	➤ Numele propriu-zis;
ExtStr	= STRING[4];	➤ Extensie.

♦ **Funcția FExpand** expandează (completează) numele cu calea. Ea este definită astfel:

FUNCTION FExpand(Nume:PathStr):PathStr

Nume este variabilă de tip PathStr care conține numele ce urmează să fie expandat cu calea. În procesul prelucrării fișierelor, funcția are sens când este utilizată pentru extinderea cu componente (unitate, cale) furnizate implicit.

Exemple: Se consideră directorul curent C:\TP

8.16. FExpand('Test1.DAT') ➤ returnează C:\TP\TEST1.DAT

8.17. FExpand('\TP\Test1.DAT') ➤ returnează C:\TP\TEST1.DAT

8.18. FExpand('\Wp51\Test1.DAT') ➤ returnează C:\WP51\TEST1.DAT

8.19. FExpand('A:TEST1.DAT') ➤ returnează A:TEST1.DAT

8.20. Programul care urmează caută un fișier într-o listă de directori. Când fișierul este găsit se afișează specificatorul extern. În caz contrar se afișează mesajul >>Fișier negăsit.

Deosebirea față de exercițiul 15 constă în aceea că, în cazul în care fișierul este găsit în directorul curent, se scrie specificatorul complet (*n:\cale\nume_fișier.extensie*).

```
PROGRAM EX20;
USES Dos;
VAR
  Nume:PathStr;
  x:STRING;
BEGIN
  Write('Numele extern al fisierului cautat (xxxxxxx.eee):');
  Readln(Nume);
  x:=FSearch(Nume, 'C:\wp51\;\wp51\rosca\;\tp\');
```

```

        IF x[0] = #0
        THEN Writeln('>>Fisier negasit')
        ELSE Writeln('>>Fisierul are specificatorul extern :
', FExpand(x))
        END.

```

♦ **Procedura FSplit** descompune specificatorul extern (format din unitate, cale, nume, extensie) în trei componente: unitate+cale, nume, .extensie. Procedura este definită astfel:

FSplit(specificator:PathStr;VAR unit_dir:DirStr;VAR name:NumeStr;VAR ext:ExtStr)

Specificator este șirul care se analizează și se descompune în componente. **Unit_dir**, **nume** și **ext** sunt variabile în care se depun cele trei componente extrase din **specificator**. Când unitatea și calea nu sunt prezente în specificator, se returnează valorile implicite. Când numele și extensia nu sunt prezente în specificator, se returnează șiruri vide.

Exemplu:

8.21. Fie un program care lucrează cu un fișier al cărui nume este introdus de la tastatură. În cazul în care de la tastatură se introduce <ENTER>, se lucrează cu fișierul implicit EX.DAT.

```

PROGRAM EX21;
USES Dos;
VAR
    f:FILE OF REAL;
    spec:PathStr; u:DirStr;
    n:NameStr; e:ExtStr;
BEGIN
    Write('Nume fișier (EX.DAT):');
    Readln(Spec);
    FSplit(Spec, u, n, e);
    IF n = '' THEN n:='Ex';
    IF e = '' THEN e:='.DAT';
    Spec:=u+n+e;
    Assign(f, spec);
    Reset(f);
    {Prelucrare}
    Close (f);
END.

```

8.7 Funcții pentru determinarea capacității și a ocupării discului

În unit-ul **Dos** există definite două funcții care returnează capacitatea unui disc și spațiul neocupat din el. Dacă funcțiile se aplică unităților de disc flexibil, în cazul în care nu este montat discul, se generează un mesaj de sistem care solicită acest lucru (afirmația este valabilă pentru toate funcțiile și procedurile Pascal care referă unități de discuri flexibile).

♦ **Funcția DiskSize** returnează capacitatea totală, în octeți, a discului montat în unitatea specificată. Ea are declarația:

Function DiskSize(Drive:BYTE):LONGINT

Drive specifică unitatea de disc, astfel: **0** - unitatea de disc curentă; **1** - unitatea de disc A; **2** - unitatea de disc B; **3** - unitatea de disc C etc.

Funcția returnează valoarea **-1** dacă unitatea de disc nu există, dacă nu este montat disc în unitate sau dacă unitatea este defectă.

♦ **Funcția DiskFree** returnează numărul de octeți liberi de pe un disc montat într-o unitate. Ea are declarația:

Function DiskFree(Drive:BYTE):LONGINT

Drive are aceeași semnificație ca la funcția **DiskSize**. Funcția returnează valoarea **-1** dacă unitatea de disc nu există, dacă nu este montat disc în unitate sau dacă unitatea este defectă.

Exemplu:

8.22. Se presupune că urmează să fie creat un fișier, MAT.DAT, care are aproximativ 10000 octeți. Se poate testa, cu o oarecare aproximare, dacă fișierul încapă pe un anumit disc (fie el **C:**). Trebuie reținut faptul că octeții liberi sunt grupați în clustere libere.

```
USES Crt;
.....
BEGIN
.....
  IF DiskFree(3) = -1 THEN
    Writeln('>>Unitatea C defecta')
  ELSE
    IF DiskFree(3) > 10000 THEN
      {Asignare, deschidere, creare}
    ELSE
      Writeln('>>Spatiu insuficient pentru fisier');
.....
```