

## Fișiere

19 mai 2006

## Lucrul cu fișiere (stdio.h)

Ca *utilizatori*, de calculatoare, ne referim la un fișier prin *nume*.  
Ca *programatori*, ne interesează accesul la *conținutul* fișierului, un șir (flux) de octeți (engl. *stream*)  
În `stdio.h`: tipul `FILE` cu elementele necesare accesului la fișier (poziția curentă în fișier, tamponul de date, indicatori de eroare și EOF).  
În program, lucrăm cu variabile `FILE *` transmise funcțiilor pt. fișiere. (nu le dereferențiem niciodată, le folosim doar pt. a indica fișiere)  
Secvența tipică de lucru: se deschide, se prelucrează, se închide fișierul.  
Fișiere standard predefinite (deschise automat la rularea programului):  
`stdin`: fișierul standard de intrare (normal: tastatura)  
`stdout`: fișierul standard de ieșire (normal: ecranul)  
`stderr`: fișierul standard de eroare (normal: ecranul)  
(sunt *constante* de tipul `FILE *` declarate în `stdio.h`)  
De fapt, `scanf/printf` etc. fac citire/scriere (de) la `stdin/stdout`  
Obs: E bine ca mesajele de eroare să fie scrise la `stderr`, pt. a putea fi separate (prin redirectare) de mesajele normale de ieșire

## Deschiderea și închiderea fișierelor

```
FILE *fopen (const char *path, const char *mode);  
– arg. 1: numele fișierului (absolut sau față de directorul curent)  
– arg. 2: modul de deschidere; primul caracter semnifică:  
r: deschidere pentru citire (fișierul trebuie să existe)  
w, a: deschidere pt. scriere; dacă fișierul nu există, e creat;  
dacă există, e trunchiat la 0 (w) sau se adaugă la sfârșit (append, a)  
În plus, șirul de caractere pt. modul de deschidere mai poate conține:  
+ permite și celălalt mod (r/w) în plus față de cel din primul caracter  
b deschide fișierul în mod binar (implicit: în mod text)  
– returnează NULL în caz de eroare (trebuie testat !!!)  
– altfel, valoarea returnată se folosește pt. lucrul în continuare  
  
int fclose(FILE *stream);  
– scrie orice a rămas în tamponurile de date, închide fișierul  
– returnează 0 în caz de succes, EOF în caz de eroare
```

## Deschiderea, închiderea și lucrul cu fișierele

Tipar pt. lucrul cu fișiere (ex. deschis pt. citire și scriere în mod text)  

```
FILE *fp; char *name = "f.txt"; /* sau din argv[], sau solicitat */  
if (!(fp = fopen(name, "rt+"))) { /* tratează eroarea */ }  
else { /* lucrează cu fișierul */ }  
if (fclose(fp)) /* eroare la închidere */;
```

La intrarea-iesirea în mod *text* se pot petrece diverse conversii în funcție de implementare (de exemplu traducere `\n` în `\r\n` pt. DOS)  
– modul *text*: doar pt. fișiere cu caractere tipăribile obișnuite, `\t`, `\n`  
– modul *binar*: pt. toate celelalte situații (chiar și pt. fișiere text)  
(asigură corespondența exactă între conținutul scris și citit)  
Citirea și scrierea într-un fișier folosesc *aceiași indicator de poziție*, care e avansat automat de fiecare operație ⇒ trebuie re-poziționat core-spunzător indicatorul când trecem între citire și scriere în același fișier  
Pentru un fișier deschis în mod dual (cu +), nu se va cita direct după scriere fără a goli tamponurile (`fflush`) sau a re-poziționa indicatorul; nu se scrie direct după citire fără re-poziționarea indicatorului sau EOF

## Citire/scriere (d)in fișiere

Cu funcții echivalente celor folosite până acum:  

```
int fputc(int c, FILE *stream); /* scrie caracter în fișier */  
int fgetc(FILE *stream); /* citește caracter din fișier */  
/*getc,putc: la fel ca și fgetc, fputc, dar sunt macrouri */  
int ungetc(int c, FILE *stream); /* pune caracterul c înapoi */  
  
int fscanf (FILE *stream, const char *format, ...);  
int fprintf(FILE *stream, const char *format, ...);  
  
int fputs(const char *s, FILE *stream); /* scrie un șir */  
int puts(const char *s); /* scrie șirul și apoi \n la ieșire */  
char *fgets(char *s, int size, FILE *stream);  
– citește până la (inclusiv) linie nouă, sau max. size – 1 caractere,  
adaugă '\0' la sfârșit ⇒ citirea sigură a unei linii, fără depășire  
returnează NULL dacă apare EOF înainte de a fi citit ceva  
  
NU FOLOSIȚI niciodată funcția gets(), nu e protejată la depășire!
```

## Exemplu: afișarea unor fișiere

```
#include <stdio.h>  
void cat(FILE *fi)  
/* afișează un fișier deschis caracter cu caracter */  
{ int c; while ((c = fgetc(fi)) != EOF) putchar(c); }  
  
void main(int argc, char *argv[]) {  
FILE *fp;  
if (argc == 1) cat(stdin); /* citește de la intrare */  
else while (--argc > 0) { /* pt. fiecare argument */  
if (!(fp = fopen(++argv, "r"))) /* deschide, testează */  
fprintf(stderr, "can't open %s", *argv);  
else { cat(fp); fclose(fp); } /* afișează, închide */  
}  
}
```

```
void clearerr(FILE *stream);
resetează indicatorii de sfârșit de fișier și eroare pentru fișierul dat

int feof(FILE *stream); /* != 0: ajuns la sfârșit de fișier */
int ferror(FILE *stream); /* != 0 la eroare pt. acel fișier */
```

#### Coduri de eroare

Dacă un apel de sistem a rezultat în eroare, se poate citi codul erorii din variabila globală extern `int errno`; declarată în `errno.h`

Se poate folosi împreună cu funcția `char *strerror(int errnum)`; din `string.h` care returnează un șir de caractere cu descrierea erorii

Se poate folosi direct funcția `void perror(const char *s); /*stdio.h*/` care tipărește mesajul s dat de utilizator, un : și apoi descrierea erorii `void exit(int status); /*stdlib.h*/` termină *normal* execuția prog.

– se scriu tamponanele, se închid fișierele, se șterg cele temporare  
– se returnează sistemului de operare codul întreg dat (v. `int main()`)

Până acum: funcții orientate pe caractere, linii, formatare (fișiere text)  
Pentru a citi/scrie un număr de octeți, neinterpretati (în format *binar*):  
`size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);`  
`size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *stream);`  
/\* citește/scrie nmemb obiecte de câte size octeți \*/

Funcțiile întorc numărul obiectelor *complete* citite/scrise corect.  
Dacă e mai mic decât cel dat, cauza se află din `feof` și `ferror`

Cu ele, putem să ne scriem funcții proprii pentru fiecare tip de date:  
`size_t readint(int *pn, FILE *stream) /* în format binar */`  
{ return fread(pn, sizeof(int), 1, stream); }  
`size_t writedbl(double x, FILE *stream) /* în format binar */`  
{ return fwrite(&x, sizeof(double), 1, stream); }

**Atenție!** `fprintf(fp, "%d", n)`; scrie întregul ca șir de cifre zecimale  
cu `fwrite` se scrie întregul în format binar (`sizeof(int)` octeți).

```
#include <errno.h>
#include <stdio.h>
#define MAX 512 /* copiem câte un sector odată */
int filecopy(FILE *fi, FILE *fo) {
    char buf[MAX];
    int size; /* nr. octeți citați */
    while (!feof(fi)) {
        size = fread(buf, 1, MAX, fi); /* citește MAX octeți */
        fwrite(buf, 1, size, fo); /* scrie doar câți s-au citit */
        if (ferror(fi) || ferror(fo)) return errno;
    }
    return 0;
}
```

```
void main(int argc, char *argv[])
{
    FILE *fi, *fo;
    if (argc != 3) {
        fprintf(stderr, "usage: copy source destination\n"); exit(1);
    } else {
        if (!(fi = fopen(argv[1], "rb"))) {
            fprintf(stderr, "%s: can't open %s: ", argv[0], argv[1]);
            perror(NULL); /* am scris deja mesajul */; exit(errno);
        }
        if (!(fo = fopen(argv[2], "wb"))) {
            fprintf(stderr, "%s: can't open %s: ", argv[0], argv[2]);
            perror(NULL); exit(errno);
        }
        if (filecopy(fi, fo)) perror("Eroare la copiere");
        if (fclose(fi) | fclose(fo)) perror("Eroare la închidere");
    }
}
```

Pe lângă citire/scriere secvențială, e posibilă poziționarea în fișier:  
`long ftell(FILE *stream); /* pozitia de la începutul fișierului */`  
`int fseek(FILE *stream, long offset, int whence); /* poziționare */`  
Al treilea parametru: punctul de referință pt. poziționarea cu `offset`:  
`SEEK_SET` (început), `SEEK_CUR` (punctul curent), `SEEK_END` (sfârșit)

`void rewind(FILE *stream); /* re-poziționează indicatorul la început */`  
(echivalent cu `(void)fseek(stream, 0L, SEEK_SET)`), plus `clearerr`

Repoziționarea trebuie efectuată:

– când dorim să "sărim" peste o anumită porțiune din fișier  
– când fișierul a fost scris, și apoi dorim să revenim să citim din el

`int fflush(FILE *stream);`  
scrie în fișier tamponanele de date nescrise pt. fluxul de ieșire `stream`

Funcțiile de tipul `printf/scanf` pot avea ca sursă/dest. și șiruri de char.  
`int sprintf(char *s, const char *format, ...);`  
`int sscanf(const char *s, const char *format, ...);`

Pentru `sprintf`, poate apărea problema depășirii tabloului în care se scrie, dacă acesta nu e dimensionat corect (suficient). Se recomandă:  
`int snprintf(char *str, size_t size, const char *format, ...);`  
în care scrierea e limitată la size caractere ⇒ variantă sigură

Între funcții similare, trebuie alese cele corespunzătoare situației. Ex:

```
int n, r; char *s, *end;
n = atoi(s); /* dacă suntem siguri; nu semnaleză erori */
n = strtol(s, &end, 10); /* se pot testa erori (s == end) și
prelucra mai departe de la end */
r = sscanf(s, "%d", &n); /* se pot testa erori (r != 1)
dar punctul de oprire în s nu e explicit (eventual cu %n) */
```