Programarea calculatoarelor

# Fișiere

5 mai 2009

# Lucrul cu fișiere

La nivel de *utilizator*, ne referim la un fișier prin *nume*.
La nivelul *interfeței de programare*, bibliotecile limbajului C definesc un tip `FILE` cu elementele necesare accesului la fișier
(poziția curentă în fișier, tamponul de date, indicatori de eroare și `EOF`)
În programe se folosesc doar pointeri la acest tip: `FILE *`.

Din punct de vedere logic, un fișier e un flux (*stream*) de octeți

Lucrul tipic cu fișiere: se deschide, se prelucrează, se închide

Fișiere standard predefinite (și deschise automat la rulare)
`stdin`: fișierul standard de intrare (normal: tastatura)
`stdout`: fișierul standard de ieșire (normal: ecranul)
`stderr`: fișierul standard de eroare (normal: ecranul)

Obs: E bine ca mesajele de eroare să fie scrise la `stderr`, pentru 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 (șir): numele fișierului (absolut sau față de directorul curent)

arg. 2 (șir): modul de deschidere; primul caracter semnifică:

`fopen` returnează `NULL` în caz de eroare (trebuie testat !!!)

Altfel, valoarea returnată (un `FILE*`) e folosită pt. lucrul în continuare

`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 (`a`, append)

Șirul de caractere pt. modul de deschidere mai poate conține apoi:

`+` permite și celălalt mod (`r/w`) în plus față de cel din primul caracter

`b` deschide fișierul în mod binar (implicit: în mod text)

```
int fclose(FILE *stream);
```

Scrie orice a rămas în tampoanele de date, închide fișierul

Returnează 0 în caz de succes, `EOF` în caz de eroare

# Deschiderea, închiderea şi lucrul cu fişierele

Secvenţa tipică de lucru cu un fişier (ex. pt. citire)

```
FILE *fp; char *name = "f.txt"; // sau cu nume din argv[], sau citit
if (!(fp = fopen(name, "r"))) { /* tratează eroarea */ }
else { /* lucrează cu fişierul */ }
if (fclose(fp)) /* eroare la închidere, trateaz-o */;
```

La intrarea/ieşirea în mod *text* se pot petrece diverse conversii în funcţie de implementare (de exemplu traducere `\n` în `\r\n` pt. DOS)
Datele citite corespund celor scrise doar dacă: toate caracterele sunt tipăribile, `\t` sau `\n`; `\n` nu e precedat de spaţii; ultimul caracter e `\n`
$\Rightarrow$ pentru orice alte situaţii, deschideţi fişierele în mod binar (asigură corespondenţa exactă între conţinutul scris şi citit)

Citirea şi scrierea într-un fişier folosesc acelaşi indicator de poziţie $\Rightarrow$ Pentru un fişier deschis în mod dual (cu +), nu se va citi direct după scriere fără a goli tampoanele (`fflush`) sau a repoziţiona indicatorul; nu se scrie direct după citire fără repoziţionarea indicatorului sau `EOF`

# Citire/scriere (d)in fişiere

Cu funcţii echivalente celor folosite până acum:

```
int fputc(int c, FILE *stream);  // scrie caracter în fişier
int fgetc(FILE *stream);         // citeşte caracter din fişier
 // getc, putc: ca şi fgetc, fputc, dar sunt macrouri (\#define)
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 şir
int puts(const char *s); // scrie şirul şi apoi \n la ieşire
char *fgets(char *s, int size, FILE *stream);
```

– citeşte până la (inclusiv) linie nouă, sau max. `size - 1` caractere, adaugă `'\0'` la sfârşit $\Rightarrow$ citirea sigură a unei linii, fără depăşire

*NU FOLOSIŢI niciodată funcţia* `gets()`*, nu e protejată la depăsire!*

# Exemplu: afişarea unor fişiere

```
#include <stdio.h>
void cat(FILE *fi) // afişează un fişier
{ int c; while ((c = fgetc(fi)) != EOF) putchar(c); }

void main(int argc, char *argv[]) {
  FILE *fp;
  if (argc == 1) cat(stdin); // fără arg, citeşte de la intrare
  else while (--argc > 0) { // pt. fiecare argument pe rând
    if (!(fp = fopen(*++argv, "r")))
      fprintf(stderr, "can't open %s", *argv);
    else { cat(fp); fclose(fp); }
  }
}
```

# Funcţii de eroare

`void clearerr(FILE *stream);`
resetează indicatorii de sfârşit de fişier şi eroare pentru fişierul dat

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

`void exit(int status);` termină execuţia programului cu val. dată

## 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 funcţia `char *strerror(int errnum);` din `string.h` care returnează un şir de caractere cu descrierea erorii

Se poate folosi direct funcţia `void perror(const char *s); // stdio.h` care tipăreşte mesajul `s` dat de utilizator, un `:` şi apoi descrierea erorii

# Citire şi scrierea în format binar

Până acum: funcţii orientate pe caractere, linii, formatare (fişiere text)
Pentru a citi/scrie direct un număr dat de octeţi, neinterpretaţi:

```
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);
```

citesc/scriu `nmemb` obiecte de câte `size` octeţi

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

În C, nu există fişiere de anumite tipuri (`file of` din PASCAL);
putem însă defini astfel de funcţii pentru fiecare tip în parte:

```
size_t readint(int *pn, FILE *stream) // intreg în format binar
{ return fread(pn, sizeof(int), 1, stream); }
size_t writedbl(double x, FILE *stream) // real în format binar
{ return fwrite(&x, sizeof(double), 1, stream); }
```

# Exemplu: copierea a două fişiere

```
#include <errno.h>
#include <stdio.h>
#define MAX 512
int filecopy(FILE *fi, FILE *fo) {
  char buf[MAX];
  int size;  // nr. octeţi citiţi
  while (!feof(fi)) {
    size = fread(buf, 1, MAX, fi);
    fwrite(buf, 1, size, fo); // scrie doar size octeţi
    if (ferror(fi) || ferror(fo)) return errno;
  }
  return 0;
}
```

# Exemplu: copierea a două fișiere (cont.)

```
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], "r"))) {
      fprintf(stderr, "%s: can't open %s: ", argv[0], argv[1]);
      perror(NULL); /* am scris deja mesajul */; exit(errno);
    }
    if (!(fo = fopen(argv[2], "w"))) {
      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");
  }
}
```

# Funcţii de poziţionare, etc.

Pe lângă citire/scriere secvenţială, e posibilă poziţionarea în fişier:

```
long ftell(FILE *stream); // pozitia de la începutul fişierului
int fseek(FILE *stream, long offset, int whence); // poziţionare
```

Al treilea parametru: punctul de referinţă pt. poziţionarea cu `offset`: `SEEK_SET` (început), `SEEK_CUR` (punctul curent), `SEEK_END` (sfârşit)

```
void rewind(FILE *stream); /* repoziţionează indicatorul la început */
```

(echivalent cu `(void)fseek(stream, 0L, SEEK_SET)`, plus `clearerr`

Repoziţionarea trebuie efectuată:
– când dorim sa "sărim" peste o anumită porţiune din fişier
– când fişierul a fost scris, şi apoi dorim să revenim să citim din el

```
int fflush(FILE *stream);
```

scrie în fişier tampoanele de date nescrise pt. fluxul de ieşire `stream`

# Funcţii de citire/scriere formatată în şiruri

Funcţiile de tipul `printf/scanf` pot avea ca sursă/dest. şi şiruri de char.

```
int sprintf(char *s, const char *format, ...);
int sscanf(const char *s, const char *format, ...);
```

Pentru `sprintf`, poate apărea problema depăşirii 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 $\Rightarrow$ variantă sigură

Între funcţii similare, trebuie alese cele corespunzătoare situaţiei. Ex:

```
int n, r; char *s, *end;
n = atoi(s); // doar când s e bun; nu semnalează erori
n = strtol(s, &end, 10); /* se pot testa erori (s == end) şi
                            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) */
```