## Lucrul cu fişiere (stdio.h)

Ca *utilizatori*, de calculatoare, ne referim la un fişier prin *nume*.
Ca *programatori*, ne interesează accesul la *conţinutul* fişierului, un şir (flux) de octeţi (engl. *stream*)
În `stdio.h`: tipul `FILE` cu elementele necesare accesului la fişier (poziţia curentă în fişier, tamponul de date, indicatori de eroare şi `EOF`).

În program, lucrăm cu variabile `FILE *` transmise funcţiilor pt. fişiere. (nu le dereferenţiem niciodată, le folosim doar pt. a indica fişiere)

Secvenţa tipică de lucru: se deschide, se prelucrează, se închide fişierul.

Fişiere standard predefinite (deschise automat la rularea programului):
`stdin`: fişierul standard de intrare (normal: tastatura)
`stdout`: fişierul standard de ieşire (normal: ecranul)
`stderr`: fişierul 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 ieşire





## Deschiderea şi închiderea fişierelor

`FILE *fopen (const char *path, const char *mode);`
– arg. 1: numele fişierului (absolut sau faţă de directorul curent)
– arg. 2: modul de deschidere; primul caracter semnifică:
`r`: deschidere pentru citire (fişierul trebuie să existe)
`w`, `a`: deschidere pt. scriere; dacă fişierul nu există, e creat; dacă există, e trunchiat la 0 (`w`) sau se adaugă la sfârşit (append, `a`)
În plus, şirul de caractere pt. modul de deschidere mai poate conţine:
`+` permite şi celălalt mod (`r/w`) în plus faţă de cel din primul caracter
`b` deschide fişierul în mod *binar* (implicit: în mod *text*)
– returnează `NULL` în caz de eroare (trebuie testat !!!)
– altfel, valoarea returnată se foloseşte pt. lucrul în continuare

`int fclose(FILE *stream);`
– scrie orice a rămas în tampoanele de date, închide fişierul
– returnează 0 în caz de succes, `EOF` în caz de eroare





## Deschiderea, închiderea şi lucrul cu fişierele

Tipar pt. lucrul cu fişiere (ex. deschis pt. citire şi 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 fişierul */ }
if (fclose(fp)) /* eroare la închidere */;
```

La intrarea-ieşirea în mod *text* se pot petrece diverse conversii în funcţie de implementare (de exemplu traducere `\n` în `\r\n` pt. DOS)
– modul *text*: doar pt. fişiere cu caractere tipăribile obişnuite , `\t`, `\n`
– modul *binar*: pt. toate celelalte situaţii (chiar şi pt. fişiere text)
(asigură corespondenţa exactă între conţinutul scris şi citit)

Citirea şi scrierea într-un fişier folosesc *acelaşi indicator de poziţie*, care e avansat automat de fiecare operaţie ⇒ trebuie repoziţionat corespunzător indicatorul când trecem între citire şi scriere în acelaşi fişier

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: la fel ca şi 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 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 ⇒ citirea sigură a unei linii, fără depăşire
returnează `NULL` dacă apare `EOF` înainte de a fi citit ceva

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





## Exemplu: afişarea unor fişiere

```
#include <stdio.h>
void cat(FILE *fi)
/* afişează un fişier 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); /* citeşte 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); } /* afişează, închide */
  }
}
```





## Funcţii de eroare

`void clearerr(FILE *stream);`
reseteză 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 */
```

### 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

`void exit(int status);/*stdlib.h*/` termină *normal* execuţia prog.
– se scriu tampoanele, se închid fişierele, se şterg cele temporare
– se returnează sistemului de operare codul întreg dat (v. `int main()`

## Citire şi scrierea directă

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

Cu ele, putem să ne scriem funcţii 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); }
```

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





## Exemplu: copierea a două fişiere

```
#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. octeţi citiţi */
  while (!feof(fi)) {
    size = fread(buf, 1, MAX, fi);    /* citeşte MAX octeţi */
    fwrite(buf, 1, size, fo); /* scrie doar câti s-au citit */
    if (ferror(fi) || ferror(fo)) return errno;
  }
  return 0;
}
```





## Exemplu: copierea a două fişiere (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`





## Alte funcţii de intrare/ieşire

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 ⇒ variantă sigură

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

```
int n, r; char *s, *end;
n = atoi(s); /* dacă suntem siguri; 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) */
```





## Preprocesorul C

– extensii (macro-uri) pentru scrierea mai concisă a programelor
– preprocesorul efectuează transformarea intr-un program C propriu-zis
– directivele de preprocesare au caracterul `#` la început de linie

`#include <numefisier>` sau `#include "numefisier"`
– include textual fişierul numit (în mod tipic definiţii)
(a doua variantă: caută întâi în directorul curent apoi în cele standard)

```
#define LEN 20  /* preprocesorul înlocuieşte LEN cu 20 peste tot */
int tab[LEN];  /* programul trebuie modificat într-un singur loc */
for (i=0; i<LEN; ++i) {/*...*/}/* codul e mai uşor de întreţinut */
```

forma generală: `#define nume(arg1,...,argn) substituţie`

```
#define max(A, B) ((A) > (B) ? (A) : (B))
#define swapint(a, b) { int tmp; tmp = a; a = b; b = tmp; }
```

*Substituţia se face textual* fără interpretare ⇒ pot apărea probleme
– folosiţi paranteze în jurul argumentelor (evită erori de precedenţă)
– argumentele: evaluate la fiecare apariţie textuală (ex. de 2x în max)
⇒ rezultat incorect la evaluarea repetată a expresiilor cu efect lateral