

Programarea calculatoarelor

Fișiere

5 mai 2009

Fișiere	3
Deschiderea și închiderea fișierelor	

```
FILE *fopen (const char *path, const char *mode);
arg. 1 (sir): numele fișierului (absolut sau față de directorul curent)
arg. 2 (sir): 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 (x) sau se adaugă la sfârșit (a, append)
Sirul 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 tamponaiele de date, închide fișierul
Returnează 0 în caz de succes, EOF în caz de eroare
```

Fișiere	5
Citire/scriere (d) în 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: ca și fgetc, fputc, dar sunt macro-uri (\#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 fputc(const char *s, FILE *stream); // scrie un șir
int fputs(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
```

NU FOLOSIȚI niciodată funcția gets(), nu e protejată la depășire!

Fișiere	2
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 pointerii 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 redirecțare) de mesajele normale de ieșire

Fișiere	4
Deschiderea, închiderea și lucrul cu fișierele	

Secvența tipică de lucru cu un fișier (ex. pt. citire)

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

La intrarea/ieșirea în mod *text* se pot petrece diverse conversii în funcție de implementare (de exemplu traducere \n în \r\n pt. DOS)
Datele citite corespund celor scrise doar dacă: toate caracterele sunt tipabile, \t sau \n; \n nu e precedat de spații; ultimul caracter e \n ⇒ pentru orice alte situații, deschideți fișierele în mod binar (sigură corespondența exactă între conținutul scris și citit)

Citirea și scrierea într-un fișier folosesc același indicator de poziție ⇒ Pentru un fișier deschis în mod dual (cu +), nu se va citi direct după scriere fără a goli tamponaiele (fflush) sau a re poziționa indicatorul; nu se scrie direct după citire fără re poziționarea indicatorului sau EOF

Fișiere	6
Exemplu: afișarea unor fișiere	

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

void main(int argc, char *argv[]) {
FILE *fp;
if (argc == 1) cat(stdin); // fără arg, citește 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); }
}
}
```

Funcții de eroare

7

```
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
void exit(int status);   termină execuția 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 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

Programarea calculatoarelor. Curs 10

Marius Mînea

Citire și scrierea în format binar

8

Până acum: funcții orientate pe caractere, linii, formatare (fișiere text) Pentru a citi/scrie direct un număr dat de octeți, neînțeptrăți:

```
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);
ctesc/scriu nmemb obiecte de câte size octeți
```

Funcțiile întorc *numărul* obiectelor *complete* citite/scrise corect.

Dacă e mai mic decât cel dat, cauza sa află din feof și ferror în C, nu există fișiere de anumite tipuri (file of din PASCAL):

putem însă defini astfel de funcții pentru fiecare tip în parte:

```
size_t readint(int *pn, FILE *stream) // întreg 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); }
```

Programarea calculatoarelor. Curs 10

Marius Mînea

Exemplu: copierea a două fișiere

9

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

Programarea calculatoarelor. Curs 10

Marius Mînea

Exemplu: copierea a două fișiere (cont.)

10

```
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 (filcopy(fi, fo)) perror("Eroare la copiere");
        if (fclose(fi) | fclose(fo)) perror("Eroare la inchidere");
    }
}
```

Programarea calculatoarelor. Curs 10

Marius Mînea

Funcții de poziționare, etc.

11

Pe lângă citire/scriere secvențială, e posibilă poziționarea în fișier: long feell(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 (inceput), SEEK_CUR (punctul curent), SEEK_END (sfârșit)

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

Repoziționarea trebuie efectuată:

- Când dorim sa "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 tamponașele de date nescrise pt. fluxul de ieșire stream

Programarea calculatoarelor. Curs 10

Marius Mînea

Funcții de citire/scriere formatată în șiruri

12

Funcțiile de tipul printf/scanf pot avea ca sursă/dest. și șiruri de char.

```
int sprintf(char *s, const char *format, ...);
int scanf(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 sprintf(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, *and;
n = atoi(s); // doar când s e bun; nu semnalează erori
n = strtol(s, &end, 10); /* se pot testa erori (s == end) și prelucra mai departe de la end */
r = scanf(s, "%d", &n); /* se pot testa erori (r != 1) dar punctul de oprire în s nu e explicit (eventual cu %n) */
```

Programarea calculatoarelor. Curs 10

Marius Mînea