Limbaje de programare

Fișiere

3 decembrie 2009

Fișiere în limbajul C

Ca *utilizatori* de calculator ne referim la un fișier prin *nume*.

În program, fișierul e o *secvență de octeți* (care se pot citi/scrie).

`stdio.h` definește tipul pointer `FILE *` pentru funcțiile cu fișiere.

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)

Obs: E bine ca mesajele de eroare să fie scrise la `stderr`,
pentru a le putea separa de scrierea la ieșirea `stdout`

Din linia de comandă, putem *redirecta* intrarea/ieșirea programului în/din fișierele dorite: `program < fis_in.txt > fis_out.txt`

Deschiderea și închiderea fișierelor

`FILE *fopen (const char *path, const char *mode);`

arg. 1: *șir* cu *numele fișierului* (absolut sau față de directorul curent)

arg. 2: *șir* reprezentând *modul de deschidere*. Primul caracter este:

`r`: deschidere pentru citire (fișierul trebuie să existe)

`w`, `a`: deschidere pt. scriere; dacă nu există, e creat;

dacă există, e trunchiat (șters) (`w`) sau se adaugă la sfârșit (`a`, append) Arg. 2 poate conține caractere suplimentare:

`+` permite modul complementar (`w/r`) primului caracter `r/w,a`

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

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

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

`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

Citire/scriere (d)in fişiere

Câte un *caracter*

```
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
```

Citire/scriere *formatată*

```
int fscanf (FILE *stream, const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
```

Câte o *linie de text*

```
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); // o ştim deja
```

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

Lucrul cu fişierele

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

```
FILE *fp;
if (!(fp = fopen("nume.ext", "r"))) { /* tratează eroarea */ }
else { /* foloseşte: getc/putc/fscanf/fgets/etc. */ }
if (fclose(fp)) /* eroare la închidere, trateaz-o */;
```

Dacă numele fişierului e dat pe linia de comandă (ex. primul arg.)

```
int main(int argc, char *argv[])
{
  FILE *f;
  if (argc != 2) { /* folosire incorecta, mesaj + terminare */ }
  else if (!(f = fopen(argv[1], "r"))) {
    /* trateaza eroarea, termina programul */ }
  }
  // foloseste fisierul, apoi inchide
}
```

Detalii: conversii, citire + scriere

La intrarea/ieșirea în mod *text* pot avea loc diverse *conversii* dependente 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 spații; ultimul caracter e `\n`
$\Rightarrow$ pentru orice alte situații, deschideți fișierele în mod *binar*
(asigură corespondența exactă între conținutul scris și citit)

Citirea și scrierea în fișier folosesc *același indicator de poziție* $\Rightarrow$ Pentru un fișier deschis în mod dual (cu `+`), nu se va citi direct după scriere fără a goli tampoanele (`fflush`) sau a repoziționa indicatorul; nu se scrie direct după citire decât la `EOF` sau repoziționând indicatorul

Exemplu: afişarea unor fişiere

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

// afişează intrarea standard sau fişierele din linia de comandă
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

```
int feof(FILE *stream);   // != 0:  ajuns la sfârşit de fişier
int ferror(FILE *stream); // != 0 dacă fişierul a avut erori
```

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

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

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 și scrierea în format binar

Până acum: funcţii orientate pe caractere, linii, formatare (fișiere 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` și `ferror`

Putem defini funcţii pentru a scrie/citi numere în format binar

```
int readint_b(FILE *stream) // intreg în format binar
{ int n; fread(&n, sizeof(int), 1, stream); return n; }
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>
#include <stdlib.h>
#define MAX 512

int 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"))) { perror(argv[1]); exit(errno); }
    if (!(fo = fopen(argv[2], "w"))) { perror(argv[2]); exit(errno); }
    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)) exit(errno);
    }
  }
  if (fclose(fi) | fclose(fo)) perror("Eroare la închidere");
}
```

Poziţionarea în fişier

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

`int fseek(FILE *stream, long offset, int whence);`

Parametrul 3: 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

la fel ca `fseek(stream, 0L, SEEK_SET); clearerr(stream);`

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

Atenţie: nu e posibilă poziţionarea în orice fişier (ex. `stdin/stdout`)

`long ftell(FILE *stream);` returnează poziţia relativ la început

`int fflush(FILE *stream);`

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