Limbaje de programare

Fișiere

28 noiembrie 2011

Fișiere și redirectarea intrării / ieșirii

Un *fișier* e o secvență de date (octeți) stocată pe un mediu extern.

Programele scrise până acum:
- au citit de la intrarea standard (uzual: tastatura)
- au scris la ieșirea standard (uzual: ecranul)

Orice program de acest fel poate lucra cu fișiere, prin:
- redirectarea intrării: `program < fis_in.txt`
- redirectarea ieșirii: `program > fis_out.txt`
- redirectarea ambelor: `program < fis_in.txt > fis_out.txt`

Lucrul cu fișiere în trei pași

1. Deschiderea fișierului: `fopen`
2. Citirea / scrierea:
   `(f)getc, (f)putc, fgets, fputs, fscanf, fprintf`
3. Închiderea fișierului: `fclose`

1: `fopen`: se referă la fișier prin *nume* (șir, `char *`)
produce un pointer de tip `FILE *` spre o structură de date internă (fișierul e gata de lucru)

2, 3: Restul funcțiilor: folosesc pointerul `FILE *` returnat de `fopen`
NU se mai referă la fișier prin nume

Moduri de deschidere a fișierelor

`r` (read): deschidere pentru citire (trebuie să existe)
`w` (write): deschidere pentru scriere (e șters dacă există)
`a` (append): deschidere pentru adăugare (poziționare la sfârșit, datele existente rămân)

La write, append: fișierul e creat dacă nu există

Variante: `r+`, `w+`, `a+`: fișierul se deschide în modul indicat, dar se poate folosi și operația complementară (scriere / citire)

Fișiere text și binare:
Implicit, un fișier e deschis în mod text. Prelucrăm astfel fișiere care conțin doar texte (caractere tipăribile, spații, `'\t'` și `'\n'`)

`b`: deschidere în mod binar (pentru orice alt format)

Deschiderea și închiderea fișierelor

`FILE *fopen (const char *pathname, const char *mode)`
arg. 1: *șir* cu *numele fișierului*
  poate fi absolut sau față de directorul curent
arg. 2: *șir* reprezentând *modul de deschidere*
  începe cu `r`, `w`, sau `a`
  opțional: `+` și/sau `b`

`fopen` returnează `NULL` în caz de eroare (trebuie testat !!!)
Altfel, valoarea returnată (un `FILE*`) e folosită mai departe

`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

Fișiere standard

În `stdio.h` sunt definite:
`stdin`: fișierul standard de intrare (normal: tastatura)
`stdout`: fișierul standard de ieșire (normal: ecranul)
`stderr`: fișierul standard de eroare (normal: ecranul)

Aceste fișiere sunt deschise automat la rularea programului

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

Citire/scriere (d)in fișiere

Câte un *caracter*

```
int fputc(int c, FILE *stream) // scrie caracter în fişier
int fgetc(FILE *stream)   // citeşte caracter din fişier
 // getc, putc: ca si fgetc, fputc
int ungetc(int c, FILE *stream) // pune caracter 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)
//citeşte din fişier în tabloul s, max. size caract. + '\0'
```

NU folosiți niciodată funcția `gets()`*, nu e protejată la depășire!*

Lucrul cu fișierele

Secvența tipică de lucru cu un fișier (exemplu pentru citire)

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

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

```
int main(int argc, char *argv[])
{
  FILE *f;
  if (argc != 2) {
    fprintf(stderr, "folosire corectă: program numefisier\n");
    return 1; // sau alt cod de eroare
  } 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* au loc *conversii* dependente de implementare (ex.: traducere `\n` în `\r\n` pt. DOS/Windows)

⇒ Fișierele care nu conțin doar text trebuie deschise în mod *binar* (asigură corespondența exactă între conținutul scris și citit)

Citirea și scrierea în fișier folosesc *același indicator de poziție*
⇒ Citiți după scriere: doar golind întâi tampoanele (`fflush`) sau repoziționând indicatorul (`fseek`)
⇒ Scrieți după citire: doar la `EOF` sau repoziționând indicatorul

Exemplu: afișarea unor fișiere

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

// afişează stdin 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); }
  }
}
```

Funcții de eroare

`int feof(FILE *stream)` != 0: ajuns la sfârșit de fișier
`int ferror(FILE *stream)` != 0 dacă fișierul a avut erori

`void exit(int status` termină execuția programului cu val. dată

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

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

Sau: funcția `char *strerror(int errnum)` din `string.h` returnează un șir de caractere cu descrierea erorii

Sau: funcția `void perror(const char *s)` din `stdio.h` tipărește mesajul `s` dat de utilizator, un `:` și apoi descrierea erorii

Citire și scrierea în format binar

Până acum am folosit funcții pentru citirea în mod text
Pentru a citi/scrie direct un număr dat de octeți, neinterpretați:

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *strm)
size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *strm)
```

citesc/scriu `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`

Putem defini funcții 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 fișiere

```
#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 dest\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");
}
```

Poziționarea în fișier

Pe lângă citire/scriere secvențială, e posibilă poziționarea în fișier:
`int fseek(FILE *stream, long offset, int whence)`
Parametrul 3: punctul de referință pt. poziționarea cu `offset`:
`SEEK_SET` (început), `SEEK_CUR` (punctul curent), `SEEK_END` (sfârșit)

`void rewind(FILE *stream)` repoziționează indicatorul la început
la fel ca `fseek(stream, 0L, SEEK_SET); clearerr(stream);`

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

NU e posibilă poziționarea în orice fișier! (ex. `stdin/stdout`)

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

`int fflush(FILE *stream)`
scrie tampoanele de date nescrise pt. fluxul de ieșire `stream`