Fişiere – Laborator 6

Lucrul cu fişiere (stdio.h)

Ca utilizatori , de calculatoare, ne referim la un fişier prin nume.

Ca programatori , ne interesează accesul la conținutul fişierului, un şir (flux) de octeți (engl. stream)

În stdio.h: tipul FILE cu elementele necesare accesului la fişier (poziția curentă în fişier, tamponul de date, indicatori de eroare şi EOF).

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

Secventă 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 față de directorul curent)

arg. 2: modul de deschidere; primul caracter semnifică:

- r: deschidere pentru citire (fişierul trebuie s_a 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 conține:

- \+ permite şi celălalt mod (r/w) în plus față 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 funcție 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 situații (chiar şi pt. fişiere text) (asigură corespondența exactă între conținutul scris şi citit)

Citirea şi scrierea într-un fişier folosesc acelaşi indicator de poziție, care e avansat automat de fiecare operație => trebuie repoziționat 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 repoziționa indicatorul; nu se scrie direct după citire fără repoziționarea 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);
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 */

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*/ terminã normal execuţia prog.
- se scriu tampoanele, se închid fişierele, se şterg cele temporare
- se returnează sistemului de operare codul întreg dat (v. int main())

Citirea 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.

Dacã e mai mic decât cel dat, cauza se aflã din feof şi ferror
Cu ele, putem sã ne scriem funcții proprii pentru fiecare tip de date:

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

Atenție! fprintf(fp, "%d", n); scrie întregul ca şir de cifre zecimale.
Cu fwrite se scrie întregul în format binar (sizeof(int) octeți.

Exemplu: copierea a douã fişiere

```
#include <errno.h>
#include <stdio.h>
#define MAX 512/* copiem câte un sector odatã */
int filecopy(FILE *fi, FILE *fo) {
    char buf[MAX];
    int size;/* nr. octeți citiți */
    while (!feof(fi)) {
      size = fread(buf, 1, MAX, fi);/* citeşte MAX octeți */
      fwrite(buf, 1, size, fo);/* scrie doar câți s-au citit */
      if (ferror(fi) || ferror(fo)) return errno;
    }
    return 0;
}



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 inchidere");
   }
}
```

Funcții de poziționare, etc.

Pe lângã citire/scriere secvențialã, e posibilã poziționarea în fişier:

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

Al treilea parametru: punctul de referintã pt. poziționarea cu offset:
SEEK SET (început), SEEK CUR (punctul curent), SEEK END (sfârşit)

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

(echivalent cu (void)fseek(stream, 0L, SEEK SET), plus clearer
Repoziționarea trebuie efectuatã:

- când dorim sã “sãrim” peste o anumitã porțiune din fişier
- când fişierul a fost scris, şi apoi dorim sã revenim sã citim din el

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

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

Alte funcții de intrare/ieşire

Funcțiile 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 funcții similare, trebuie alese cele corespunzătoare situației. 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) _si
                           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 într-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 definiții)

(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 întreținut */
```

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

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

Substituția se face textual fără interpretare => pot apărea probleme

- folosiți paranteze în jurul argumentelor (evită erori de precedență)
- argumentele: evaluate la fiecare apariție textuală (ex. de 2x în max)
- rezultat incorect la evaluarea repetată a expresiilor cu efect lateral

Citirea de la intrare

- un cuvânt (orice până la spațiu alb)

```
char s[80]; scanf("%79s", s);
```

spațiu alb = spațiu sau \f \n \r \t \v

ignoră spații albe inițiale; adaugă '\0' la sfârşit

Atenție! Nu se poate citi o linie de text în acest fel !

din Acesta e un text va citi doar primul cuvânt: Acesta

- o linie de text, până la '\n'

```
char s[80]; fgets(s, 80, stdin);
```

citeşte max. 80-1 caractere, inclusiv '\n', adaugă '\0' la sfârşit

stdin: identificator definit în stdio.h pt. fişierul standard de intrare

Testarea sfârşitului de fişier

1. În orice punct de program: int feof(FILE *fp)

returnează nenul (adevărat) dacă s-a atins sfârşitul lui fp; 0 dacă nu pentru fişierul de intrare: feof(stdin)

2. După valoarea returnată de funcțiile de intrare:

```
int c; c = getchar(); if (c == EOF) /* ... */
```

Atenție! c trebuie declarat int pentru a testa de EOF

valoarea EOF (-1) e diferită de cea a oricărui caracter (0 .. 255)

scanf returnează EOF (-1) dacă întâlneşte imediat sfârşitul de fişier (nu şi dacă a reuşit să citească măcar ceva)

=> Folosiți doar if (scanf(...) == nr_variabile_dorite) pentru a testa citire corectă, nu doar if ((scanf(...))
(pentru că şi EOF e nenul)
fgets returnează NULL dacă fişierul se termină înainte de a citi ceva
Exemplu: prelucrarea unui fişier linie cu linie
char lin[128]; while (fgets(lin, 128, stdin)) /* prelucrează lin */

Citirea anumitor categorii de caractere

Atenție: Funcțiile din ctype.h returnează nenul pentru un character de felul dorit şi 0 în caz contrar (NU neapărat 1 şi 0)
nu scrieți niciodată if (isalpha(c) == 1) ci doar if (isalpha(c))
Atenție la cicluri infinite pentru sfârşit de fişier:
int c; while (isdigit(c = getchar())) /* ceva */
va ieşi din ciclu când c nu e cifră, inclusiv la EOF (nu e cifră)
int c; while (!isdigit(c = getchar())) /* ceva */
se va bloca la EOF, pentru că nu e cifră (nici isalpha, isspace, etc.)
while (!isdigit(c = getchar()))
if (c == EOF) break; /* sau ce vrem să facem la EOF */
else /* restul prelucrării */
Ignoră oricâte spații albe: scanf(" ");
Ignoră până la sfârşit de linie: scanf("%*[^\n]"); scanf("%*1[\n]");
(citeşte şi ignoră (*) oricâte caractere diferite (^) de \n şi un \n

Citirea şi prelucrarea până la sfârşit de fişier

Pentru a prelucra corect până la EOF nu e suficientă secvența din stânga deoarece !feof() la începutul ciclului nu garantează o citire corectă.
Trebuie testată citirea corectă (getchar() != EOF, fgets(...) != NULL, valorea lui (f)scanf), şi tratat cazul de eroare (ex. ieşirea din buclă)
while !(feof(fisier)){ for (;;) {
citeste();if (citeste() != CORECT) break;
prelucreaza();prelucrează();
}}
Indicatorul de sfârşit de fişier e poziționat doar când se încearcă citirea dincolo de sfârşitul fişierului, nu când s-a citit ultimul caracter.
- după citirea ultimului element, feof() poate fi adevărat sau nu
ex. pt. un fişier de întregi separați prin spații, feof() e poziționat după citirea ultimului doar dacă nu e urmat de altceva (ex. spațiu, \n)
ex. la citirea linie cu linie, feof() e poziționat după citirea ultimei linii doar dacă ea nu se termină cu \n.
- dacă feof() e fals, fişierul poate să mai conțină un element sau nu

Problemă de laborator

Să se tipărească, pe câte o linie, toate secvențele de cifre din intrare.
O abordare: textul e o repetiție de: grup de cifre, grup de alte caractere =>structura: două cicluri consecutive, într-un ciclu pana la EOF
'\n' se tipăreşte la trecerea între cele două (preferabil după cifre)
se începe cu grupul de alte caractere (posibil vid)

```
void main(void)
{ int c;
    do { /* şi EOF e !isdigit, atentie la ciclu infinit! */
    while (!isdigit(c = getchar())) if (c == EOF) return;
    /* aici, c e sigur o cifra; repeta cat timp e cifra */
    do putchar(c); while (isdigit(c = getchar()));
    putchar('\n'); /* gata cifrele, c e altceva, poate EOF */
  } while (c != EOF);
}
```

```
   Mai simplu: când vedem o cifră, citim şi tipărim cât timp e cifră
void main(void) {
  int c;
  while ((c = getchar()) != EOF)
    if (isdigit(c)) { /* prima cifra; continua cu restul */
       do putchar(c); while (isdigit(c = getchar()));
       putchar('\n'); /* gata grupul de cifre */
    } /* daca nu e cifra, nu trebuie facut nimic */
}
```

Probleme :

1. Ştiind că fişierul numere.in conține numere întregi să se afişeze pe ecran numărul elementelor pare din fişier.

2. Fişierul sir1.txt conține un şir de numere întregi. Elementele şirului se găsesc în fişier unul sub altul, fiecare pe câte un rând. Să se scrie pe un rând al fişierului sir2.txt, separate, elementele prime ale şirului dat.

3. Ştiind că în fişierul matrice.in se găseşte pe prima linie un număr care reprezintă numărul de linii şi de coloane pentru o matrice pătratică, iar pe următoarele linii matricea efectivă, să se scrie un program care să citească din fişier matricea, să o afişeze pe ecran şi să afişeze numărul elementelor pare de pe diagonala principală din matrice. Să se verifice dacă toate elementele de sub diagonala principală sunt nule.

4. Pentru evidența angajaților unei firme, se foloseşte fişierul pers.in care conține: pe primul rând numărul de angajați, apoi pe fiecare rând, codul, salariul şi numele unui angajat, în această ordine, separate prin spații. Realizați un program care scrie în fişierul pers.out datele angajatului (angajaților) cu cel mai mic salariu (fiecare angajat pe câte un rând în acelaşi format ca şi în fişierul de intrare).

5. Un fisier contine 7 linii, pe fiecare scriind o zi a saptamanii urmata dupa un spatiu de suma cheltuita in ziua respectiva. Sa se scrie un program C ce prelucreaza fisierul de mai sus si afiseaza suma totala cheltuita, ziua cu suma maxima si suma medie. Daca sunt mai multe zile cu aceeasi suma maxima, puteti tipari doar prima zi in care s-a cheltuit suma sau toate.

6. Sa se scrie un program care executa in mod repetat urmatoarele operatii:
- Preia informatiile (nume si nota) pentru o grupa de studenti. Citirea se face dintr-un fisier al carui nume se specifica de utilizator.
- Verifica prezenta unui student in grupa
- Listeaza grupa in ordine alfabetica si afiseaza media grupei.
- Termina program.

7. Sa se scrie un program de copiere a unui fisier in alt fisier, folosindu-se numai functii de prelucrare pe caractere.

8. Sa se scrie un program de copiere a unui fisier in alt fisier, folosindu-se numai functii de prelucrare functii pe siruri de caractere.