Fișiere, input și output

Input și output

- Funcțiile de bază pentru I/O sunt aflate în header-ul standard *stdio.h*
- Permite interacțiunea programelor cu mediul exterior (consolă, tastatură, fișiere, alte procese, rețeua, etc.)
- Vom discuta doar despre I/O de bază, folosind fișiere salvate pe mașina locală

Accesul la fișiere

- Pentru a interacționa cu un fișier local, *stdio.h* pune la dispoziție tipul FILE
- Acesta este o structură, implementată diferit în funcție de SO, compilator și versiunea de C folosită
- Nu este recomandat să folosiți direct câmpurile definite în această structură, nici măcar nu trebuie să le cunoașteți
- Mai mult, întotdeauna trebuie să folosiți un pointer la structură, deci FILE*

Exemple definiție FILE

```
typedef struct _iobuf
{
   char*   _ptr;
   int _cnt;
   char*   _base;
   int _flag;
   int _file;
   int _charbuf;
   int _bufsiz;
   char*   _tmpfname;
} FILE;
```

```
typedef struct  {
      int           level;      /* fill/empty level of buffer */
      unsigned      flags;      /* File status flags          */
      char          fd;         /* File descriptor            */
      unsigned char   hold;     /* Ungetc char if no buffer   */
      int           bsize;      /* Buffer size                */
  unsigned char   *buffer;    /* Data transfer buffer       */
  unsigned char   *curp;      /* Current active pointer     */
      unsigned      istemp;     /* Temporary file indicator   */
      short         token;      /* Used for validity checking */
}      FILE;
```

Exemplu FILE*

Untitled20.c

```c
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <conio.h>
#define CLOSE_FILE 1
void verify_file(FILE* f)
{
    if (f != NULL)
    {
        printf("%p\n", f);
        if (CLOSE_FILE)
            fclose(f);
    }
    else
    {
        printf("Nu am putut deschide fisierul! Eroare: %s\n", strerror(errno));
    }
}
int main()
{
    FILE *f1, *f2;
    f1 = fopen("Untitled20.c", "r");
    verify_file(f1);
    f1 = fopen("Untitled30.c", "r");
    verify_file(f1);
    f2 = fopen("Untitled18.c", "r");
    verify_file(f2);

    printf("Nr maxim fisiere deschise: %d\n", FOPEN_MAX);

    getch();
```

C:\Users\traian\Documents\Untitled20.exe

```
76492960
Nu am putut deschide fisierul! Eroare: No such file or directory
76492960
Nr maxim fisiere deschise: 20
```

Deschiderea fișierelor

- FILE * fopen (const char * filename, const char * mode);

- filename = calea către fișier
  - Relativă (față de directorul de unde se pornește programul executabil, de ex “foo.in”)
  - Absolută
    - In Linux: “/var/www/httpd/httpd.conf”
    - In Windows: “C:\Windows\windows.ini”
  - Case-sensitive – depinde de SO
- mode = specificatori de mod, specifică cum va fi deschis fișierul (text vs binar, citire vs scriere vs append)
- Întoarce un pointer FILE* la structura asociată fișierului
- FILE* mențione asocierea către *stream-ul* de I/O asociat fișierului

Deschiderea fișierelor

- Dacă stream-ul nu poate fi deschis, se întoace NULL
- Se poate afișa motivul pentru care nu poate fi deschis, folosind variabila *errno* (din *errno.h*) și funcția *strerror* (din *string.h*)
- În mod uzual, toate fișierele deschise folosesc un buffer
  - Drept urmare, scrierile și citirile folosesc un buffer de memorie pentru accesul eficient

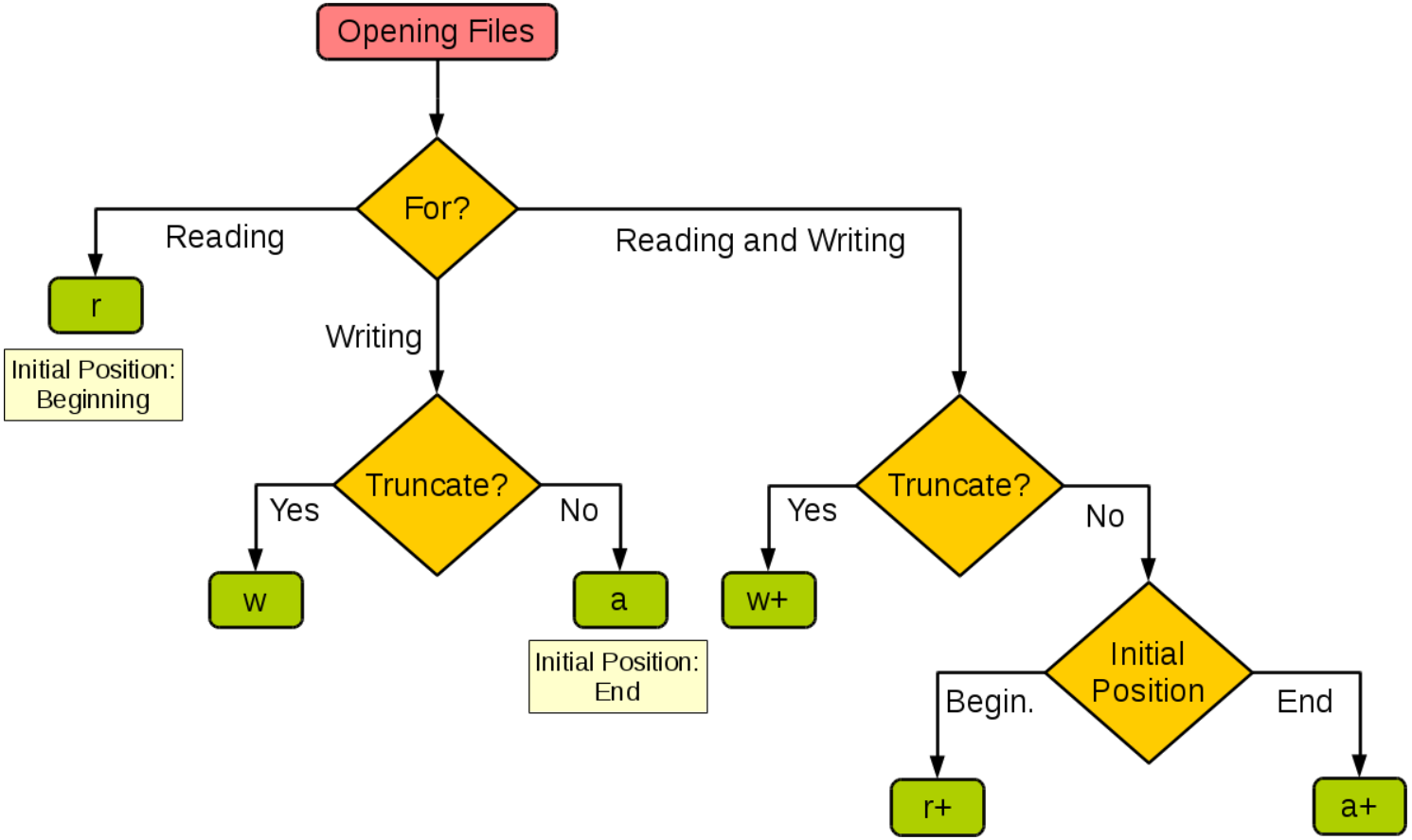
```
f = fopen("Untitled20.c", "r");
if (f != NULL)
        printf("%p\n", f);
else
        printf("Nu am putut deschide fisierul! Eroare: %s\n", strerror(errno));
```

Moduri deschidere

- “r” – citire, fișierul trebuie să existe
- “w” – scriere, creează de fiecare dată un fișier nou. Dacă acesta există deja, conținutul este ignorat.
- “a” – scriere, cu adăugare la finalul fișierului. Dacă fișierul nu există deja, este creat unul nou.
  - Stream-ul astfel deschis nu permite operații de poziționare în fișier (discutate mai târziu)
- “r+” – deschide fișier pentru update (I și O). Fișierul trebuie să existe
- “w+” – deschide un fișier gol pentru update (I și O). Dacă acesta există deja, conținutul este ignorat.
- “a+” – deschide un fișier pentru update (I și O). Operațiile de poziționare sunt permise, dar doar pentru I. Operațiile de O se fac automat la sfârșitul fișierului.

Moduri deschidere

- http://stackoverflow.com/questions/1466000/python-open-built-in-function-difference-between-modes-a-a-w-w-and-r



Moduri deschidere

- Cu specificatorii de mod de pe slide-ul anterior, stream-ul va fi deschis ca un *fișier text*
- Pentru a-l deschide ca *fișier binar (fișier de date)* trebuie adăugat caracterul “b” undeva după primul caracter din specificatorii menționați anterior
  - De exemplu: “rb”, “a+b”, “ab+”
- C2011 adaugă specificatorul “x” care poate fi folosit pentru scriere (doar împreună cu “w”). fopen va întoarce null dacă fișierul există și deschide pentru scriere doar fișiere noi.

Închiderea fișierelor

- int fclose(FILE * stream);

- Stream = pointer către fișierul care specifică stream-ul ce se dorește să fie închis
- Eliberează resursele folosite de stream (de ex. buffer-ul)
- Dacă folosește un buffer, conținutul din buffer este scris in fișier
- Dezasociază stream-ul de fișierul fizic. Chiar și dacă apelul eșuează, stream-ul va fi dezasociat de fișier.
- Returnează 0 pentru succes, EOF (de obicei, -1 sau un număr negativ) pentru insucces

Standard input și output

- Există următoarele stream-uri default definite în *stdio.h*
- FILE* stdin
  - Standard input stream
  - Pe majoritatea sistemelor, este tastatura
  - Dacă stream-ul de intrare este un dispozitiv interactiv, în general stream-ul nu are buffer asociat. Însă depinde de implementare/versiune de C.
- FILE* stdout
  - Standard output stream
  - Pe majoritatea sistemelor, este consola
  - Dacă stream-ul de ieșire nu este un dispozitiv interactiv, în general stream-ul are un buffer asociat.
- FILE* stderr
  - Standard error stream
  - Este tot un stream de ieșire, special pentru erori
  - Pe majoritatea sistemelor, este tot consola, însă poate fi redirecționat către alte fișiere/stream-uri

Redirecționare stdin, stdout

- Redirectare stdin către alt fișier de input
  - bubu < foo.in
- Redirectare stdout către alt fișier de output
  - bubu > foo.out (rewrite)
  - bubu >> foo.out (append)
- Redirectare atat stdout, cât și stderr
  - bubu &> foo.out
  - bubu > foo.out 2>&1
- Fișiere diferite
  - bubu > foo.out 2> foo.err
- Cu pipe-uri între procese
  - bubu1 | bubu2

Formatare input și output

- int printf (const char * format, ...);
- Permite afișarea formatată a variabilelor la stdout
- Format conține text normal și specificatori de format
- Întoarce numărul de caractere scrise, poate întorce un număr negativ în caz de eroare

- int scanf (const char * format, ...);
- Permite citirea variabilelor de la stdin, conform specificatorilor de format
- Sare peste whitespace
- Întoarce numărul de variabile din lista de argumente care au fost corect citite, sau eroare, sau EOF

Remember! Specificatori de format

- Din K&R
- %[flags][width][.precision][length]specifier

Character	Argument type; Printed As
d, i	int; decimal number
o	int; unsigned octal number (without a leading zero)
x, X	int; unsigned hexadecimal number (without a leading 0x or 0X), using abcdef or ABCDEF for 10, ...,15.
u	int; unsigned decimal number
c	int; single character
s	char *; print characters from the string until a '\0' or the number of characters given by the precision.
f	double; [-]*m.dddddd*, where the number of *d*'s is given by the precision (default 6).
e, E	double; [-]*m.ddddddd*e+/-*xx* or [-]*m.ddddddd*E+/-*xx*, where the number of *d*'s is given by the precision (default 6).
g, G	double; use %e or %E if the exponent is less than -4 or greater than or equal to the precision; otherwise use %f. Trailing zeros and a trailing decimal point are not printed.
p	void *; pointer (implementation-dependent representation).
%	no argument is converted; print a %

- More: http://www.cplusplus.com/reference/cstdio/printf/

Formatare input și output

- Există funcții de formatare și pentru a scrie în alte stream-uri
- int fprintf (FILE * stream, const char * format, ...);
- int fscanf (FILE * stream, const char * format, ...);

- Sau se pot scrie/citi formatat informații într-un string și apoi folosit acesta mai departe
- int sprintf (char * str, const char * format, ...);
- int sscanf (const char * s, const char * format, ...);
- int snprintf (char * s, size_t n, const char * format, ...);

Exemplu fprintf

```c
/*
fprintf example
Adapted from: http://www.cplusplus.com/reference/cstdio/fprintf/
*/
#include <stdio.h>
#include <string.h>

int main ()
{
   FILE * pFile;
   int i = 1;
   char name [100];

   pFile = fopen ("myfile.txt","w");
   while (1){
     puts ("please, enter a name: ");
     gets (name);
     if (strlen(name) == 0)
        break;
     fprintf (pFile, "Name %d [%-10.10s]\n", i++, name);
   }
   fclose (pFile);

   return 0;
}
```

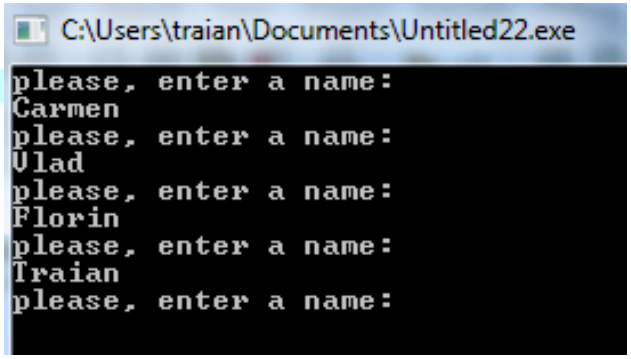
Exemplu fprintf

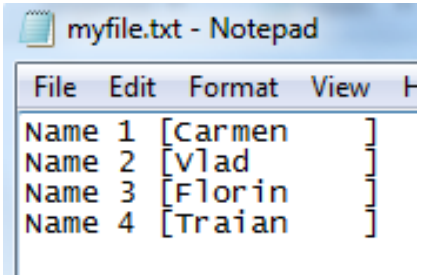
```c
/*
fprintf example
Adapted from: http://www.cplusplus.com/reference/cstdio/fprintf/
*/
#include <stdio.h>
#include <string.h>

int main ()
{
   FILE * pFile;
   int i = 1;
   char name [100];

   pFile = fopen ("myfile.txt","w");
   while (1){
     puts ("please, enter a name: ");
     gets (name);
     if (strlen(name) == 0)
        break;
     fprintf (pFile, "Name %d [%-10.10s]\n", i++, name);
   }
   fclose (pFile);

   return 0;
}
```





Buffered streams

- **Operațiile cu HDD sunt mult mai încete decât cele cu memoria (RAM). Drept urmare, se folosește un buffer pentru a citi/scrie blocuri mai mari în memorie înainte de reciti/scriere efectiv buffer-ul pe HDD**
- Toate stream-urile deschise cu fopen() folosesc buffere dacă se știe că nu se referă la un dispozitiv interactiv
- Tipuri de buffering: full buffer, line buffer, no buffer
- Bufferul se poate schimba cu:
  - void setbuf (FILE * stream, char * buffer);
  - buffer-ul trebuie să aibă cel puțin BUFSIZ octeți
- Tipul de buffering se poate schimba cu:
  - int setvbuf (FILE * stream, char * buffer, int mode, size_t size);

Buffered streams - fflush

- int fflush (FILE * stream);
- Pentru a forța scrierea buffer-ului stream-ului în fișier se poate folosi fflush()

```c
#include <stdio.h>

int main()
{
    FILE *f = fopen("myfile2.txt", "w");
    if (f){
        fputc('a', f);
        sleep(10000);
        fflush(f);
        sleep(10000);
        fclose(f);
    }
    return 1;
}
```

Fișiere text vs fișiere binare

- Fișiere text (*plain text*)
  - Tip special de fișiere care conține doar text (șiruri de caractere) și separatori de linii
  - Nu poate conține formatare (bold, italic, etc.) sau date speciale
- Fișiere binare / de date
  - Conțin un amestec de text și date, sau doar date
  - La extrem, orice program de calculator poate fi văzut ca un fișier binar
  - Dar și alte fișiere sunt binare, deși conțin și text

Fișiere text vs fișiere binare

- Care dintre următoarele tipuri de fișiere sunt text și care sunt binare?
- .txt , .doc , .c , .cpp , .xsl , .mp3 , .mov , .srt , .jpeg , .html , .ini , .dll , .exe
- MIME types: https://en.wikipedia.org/wiki/MIME#Content-Type pentru generalizare

Fișiere text vs fișiere binare

```
TPlayer* alloc(int nMaxUnits)
{
    static int id=0;//nu e necesara initializarea cu 0
    TPlayer *p=malloc(sizeof(TPlayer));
    if (!p)
        return NULL;
    p->units=malloc(nMaxUnits*sizeof(TUnit));
    if (!p->units)
    {
        free(p);
        return NULL;
    }
    p->id=id++;
    p->nMaxUnits=nMaxUnits;
    p->nCrtUnits=0;
    return p;
}
```

Fișiere text

- Primele fișiere
- Conțin doar șiruri de caractere, separate pe linii
- Nu pot conține în mod direct alte variabile (în afară de text) decât dacă sunt transformate în string-uri înainte (de exemplu toString())
- C-ul standard știe să citească doar fișiere text ASCII
- Este mai dificil de citit fișiere care au alt *encoding* al textului însă există funcții externe care pot ajuta
- Majoritatea limbajelor de programare au pachete speciale pentru a citi fișiere text cu orice *encoding* standard (de ex. UTF-8)

Citire / scriere fișiere text

- Citire / scriere caracter cu caracter
- Citire / scriere linie cu linie
- Citire / scriere formatată

Citire / scriere caracter cu caracter

- int fgetc (FILE * stream);
- Întoarce caracterul curent din fișier. După aceea, indicatorul de poziție din fișier este avansat la următorul caracter
- Întoarce EOF dacă s-a ajuns la final

- int fputc (int character, FILE * stream);
- Scrie caracterul la poziția la care se află indicatorul de poziție. După aceea, indicatorul de poziție din fișier este incrementat +1.
- Întoarce caracterul scris sau EOF

Citire / scriere linie cu linie

- char * fgets (char * str, int num, FILE * stream);
- Citește *num-1* caractere din stream sau până la întâlnirea *newline* sau EOF
- Întoarce str dacă citirea a fost făcută cu succes
- Dacă citirea nu este făcută cu succes, se poziționează indicatorul de eof (accesibil cu *feof()*), iar în unele cazuri se întoarce NULL

- int fputs (const char * str, FILE * stream);
- Scrie string-ul din *str* în *stream*, mai puțin terminatorul de string și adaugă un *newline* după aceea în fișier
- Întoarce valoare non-negativă pentru succes, EOF altfel

Fișiere binare

- Conțin atât text, cât și date împachetate binar, sau doar date binare
- Nu sunt vizualizate ușor cu un editor text uzual (notepad, notepad++, sublime, vi, gedit, etc.)
- Folosite de majoritatea aplicațiilor complexe pentru a salva datele
- Există și conceptul de serializare a variabilelor/obiectelor pentru a le reține valorile între două rulări diferite ale programului
  - În C nu există o soluție oficială pentru serializare: http://stackoverflow.com/questions/6002528/c-serialization-techniques
  - În alte limbaje (de ex. Java), există metode specifice pentru aceste operații de I/O pentru o anumită variabilă

Citire / scriere fișiere binare

- size_t fread (void * ptr, size_t size, size_t count, FILE * stream);
- Citește un vector de *count* elemente din *stream*, fiecare element având dimensiunea *size*, și le salvează în zona de memorie specificată de *ptr*
- Întoarce numărul de elemente citite. Dacă acesta este mai mic decât count, atunci ori s-a ajuns la EOF (se verifică cu *feof()*), ori a fost întâmpinată o eroare (se verifica cu *ferror()*).

- size_t fwrite (const void * ptr, size_t size, size_t count, FILE * stream);
- Scrie blocul de memorie de dimensiune size * count în fișier
- Avansează indicatorul de poziție
- Întoarce același lucru ca fread()

Exemplu fwrite

- Care va fi diferența între cele două fișiere?

```c
int main()
{
    TPunct p = {3, 2, 1, "alea iacta est"};
    FILE* f = fopen("myfile3.bin", "w");
    FILE* f1 = fopen("myfile4.bin", "w");
    printf("%d", sizeof(TPunct));

    if (f)
    {
        fwrite_punct(&p, f);
        fclose(f);
    }
    if (f1)
    {
        fwrite(&p, sizeof(p), 1, f1);
        fclose(f1);
    }

    return 1;
}
```

```c
#include <stdio.h>
#include <conio.h>

typedef struct {
    double x, y;
    char active;
    char name[24];
} TPunct;

void fwrite_punct(TPunct* p, FILE* f)
{
    fwrite(&(p->x), sizeof(p->x), 1, f);
    fwrite(&(p->y), sizeof(p->y), 1, f);
    fwrite(&(p->active), sizeof(p->active), 1, f);
    fwrite(&(p->name), sizeof(p->name), 1, f);
}
```

ftell

- long int ftell (FILE * stream);
- Întoarce valoarea curentă a indicatorului de poziție în fișier
- Pentru fișiere binare: numărul de octeți de la începutul fișierului
- Pentru fișiere text: nu este garantat că este numărul de caractere de la începutul fișierului
  - For a text stream, its file position indicator contains unspecified information, usable by the fseek function for returning the file position indicator for the stream to its position at the time of the ftell call.
- Întoarce poziție non-negativă sau -1L pentru eroare

fseek

- int fseek (FILE * stream, long int offset, int origin);
- Setează indicatorul de poziție al *stream*-ului la o poziție nouă

- For streams open in binary mode, the new position is defined by adding *offset* to a reference position specified by *origin*.
- For streams open in text mode, *offset* shall either be zero or a value returned by a previous call to ftell(), and *origin* shall necessarily be SEEK_SET.
- If the function is called with other values for these arguments, support depends on the particular system and library implementation (non-portable).

- Valori posibile origin
- SEEK_SET Început fișier
- SEEK_CUR Poziție curentă în fișier
- SEEK_END Sfârșit fișier

feof, ferror

- int feof (FILE * stream);
- Verifică dacă a fost setat indicatorul de EOF
- Se poate ajunge la EOF, iar indicatorul să nu fie setat încă pentru că nu s-a făcut o încercare de citire de acolo (sau de avansare a indicatorului de poziție)
- Întoarce o valoare diferită de zero dacă s-a ajuns la EOF, 0 altfel

- int ferror (FILE * stream);
- Verifică dacă a fost setat indicatorul de eroare asociat fișierului

Exemplu modificare tag ID3v1

- ID3 este un mecanism pentru a atașa metadate despre un mp3
- Sunt mai multe versiuni, v1, v2, etc.
- Informațiile sunt adăugate în formatul standard mp3 (binar), de multe ori la început sau la sfârșit
- Aceste metadate (nume artist, melodie, album, an, etc.) sunt redate de mp3 playere și alte programe care randeaza muzică

- Modificare tag ID3v1 aflat la finalul unui fișier mp3
- Folosește structura fixă a descrierii ID3v1

```c
#include <stdio.h>

int main()
{
        FILE*f =fopen("Dillon.mp3","r+b");
        long position=0;char tag[4], title[31]="testtitle";
        if(!f)
        {
                printf("nu s-a putut deschide");
                return -1;
        }

        fseek(f,0,SEEK_END);
       position=ftell(f);
       fseek(f,position-125,SEEK_SET);
       //fread(tag,1,3,f);
       //tag[3]='\0';
       fwrite(title,1,30,f);
       fclose(f);
       //printf("tag: %s\n",tag);
}
```

Strings are either space- or zero-padded. Unset string entries are filled using an empty string. ID3v1 is 128 bytes long.

Field	Length	Description
header	3	"TAG"
title	30	30 characters of the title
artist	30	30 characters of the artist name
album	30	30 characters of the album name
year	4	A four-digit year
comment	28[4] or 30	The comment.
zero-byte[4]	1	If a track number is stored, this byte contains a binary 0.
track[4]	1	The number of the track on the album, or 0. Invalid, if previous byte is not a binary 0.
genre	1	Index in a list of genres, or 255

Comparație performanță fgetc, fgets, fread

- Studiu preluat de la adresa: http://www.nextpoint.se/?p=540
- Buffer normal

SIZE	fgetc()	fgets()	fread()
1K	0.000170	0.000045	0.000029
10K	0.001288	0.000301	0.000103
100K	0.012736	0.002904	0.000848
1M	0.120394	0.026483	0.007996
10M	1.120597	0.282562	0.080213
100M	10.798302	2.541511	0.744125
200M	21.437850	5.030052	1.488380
500M			3.819704
1G			7.494000

Buffer 4*BUFSIZ

SIZE	fgetc()	fgets()	fread()
1K	0.000147	0.000044	0.000026
10K	0.001300	0.000312	0.000103
100K	0.012643	0.002995	0.000856
1M	0.110152	0.025210	0.007382
10M	1.077510	0.256205	0.074444
100M	11.294960	2.565854	0.743703
200M	21.893663	5.197142	1.492345
500M			3.769666
1G			7.475868

Comparație performanță fread

- Același studiu: fread() vs dimensiune buffer

SIZE	256	512	1K	4K	16K	32K	1M	filesize
1K	0.000029	0.000023	0.000023	0.000025	0.000025	0.000025	0.000026	0.000029
10K	0.000121	0.000109	0.000107	0.000108	0.000098	0.000099	0.000100	0.000103
100K	0.001048	0.000985	0.000975	0.000954	0.000893	0.000878	0.000856	0.000848
1M	0.009288	0.008926	0.008762	0.008641	0.008180	0.007918	0.007758	0.007996
10M	0.096388	0.088114	0.084811	0.083801	0.078113	0.076242	0.074664	0.080213
100M	0.921991	0.971907	0.875404	0.864597	0.812085	0.818127	0.822893	0.744125
200M	1.787666	1.725685	1.696015	1.672898	1.568236	1.528346	1.495935	1.488380
500M	4.616582	4.526420	4.375277	4.318964	4.055546	3.937942	3.901319	3.819704
1G	9.031612	8.807461	8.797347	8.371005	7.826087	7.617529	7.461680	7.494000