



Universitatea "Politehnica" din București
Facultatea de Electronică, Telecomunicații și
Tehnologia Informației



Programarea Calculatoarelor (limbajul C)

Curs 8 – Lucrul cu pointeri și fișiere de date

Prof. Bogdan IONESCU

2016-2017

Cuprins

- 8.1. Lucrul cu pointeri
- 8.2. Lucrul cu fișiere de date

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017

1/66

8.1. Lucrul cu pointeri

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017

2/66

Ce reprezintă un pointer ?

- > Pentru a înțelege noțiunea de pointer vom relua în discuție noțiunea de variabilă.
- > **Variabilă** = un "container" sau o zonă de memorie în care sunt stocate anumite valori.

VarA

2009

↓

nume

→

valoarea variabilei

→

2567843

→

adresa de memorie

- > Zona de memorie nu este oarecare, ci aceasta are o anumită adresă unică (un număr) pe baza căreia se poate face referință la ea.
- > Variabila este identificată printr-un **nume** alfanumeric unic.

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017

3/66

Ce reprezintă un pointer ? (continuare)

VarA

2009

↓

nume

→

valoarea variabilei

→

2567843

→

adresa de memorie

Ce se întâmplă dacă scriem:

```
printf("%d", VarA);
```

>2009

VarA indică valoarea

Dar dacă scriem:

```
printf("%d", &VarA);
```

>2567843

&VarA indică adresa

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017

4/66

Ce reprezintă un pointer ? (continuare)

- > **Un pointer este tot o variabilă**, adică o locație de memorie.
- > Acesta diferă de variabilele clasice prin faptul că, în loc să stocheze valori, **acesta stochează doar adrese** ale locațiilor din memorie.
- > De aici și denumirea de pointer – indicator, prin intermediul adresei pe care o stochează, putem spune că acesta indică spre conținutul unei alte locații de memorie.

2009

2567843

↓

PointerA

→

valoarea pointerului

→

2567843

→

10000

→

adresa variabilei pointer

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017

5/66

Lucrul cu pointeri

> Pointerii constituie fundamentul limbajului C, aceștia oferind o multitudine de avantaje:

- **rapiditate în manipularea datelor**, exemplu: vrem să copiem valorile vectorului A în vectorul B, folosind pointerii este suficient să stocăm adresa primului element din A în adresa primului element din B,
- **eficientizare a memoriei**, variabilele definite ca pointeri pot fi alocate dinamic: se alocă **când** vrem, **cât** vrem și **cât timp** vrem (se poate elibera memoria),
- anumite calcule nu pot fi exprimate decât pe baza pointerilor,

Lucrul cu pointeri (continuare)

> Avantaje (continuare):

- **o funcție poate returna mai multe valori** cu ajutorul pointerilor, returnând astfel adresa la care se găsesc acestea (de exemplu, adresa de început a unui vector)
- **transmiterea valorilor prin adresă** (pointer) în funcții permite modificarea valorilor acestora, ceea ce nu este valabil când parametrul funcției este transmis prin valoare (cazul clasic),
- etc.

> Principalul dezavantaj este dat manipularea acestora, ce **necesită anumite precauții**, anumite greșeli de manipulare nefiind sesizate de compilator !!!

Lucrul cu pointeri (continuare)

> În limbajul C studiat până în acest punct, ați lucrat deja cu pointeri, și anume la:

- lucrul cu **vectori și matrice**,
- lucrul cu **șiruri de caractere**,
- lucrul cu **structuri de date și uniuni**,
- lucrul cu **funcții**.

> Modul de definire al unui pointer:

```
<tip de bază> *<nume pointer>;
```

> **Efect**: variabila <nume pointer> va fi un pointer ce indică o locație de memorie ce stochează valori de tip <tip de bază>.

Lucrul cu pointeri (continuare)

> Exemple:

```
int *PointerInt;  
double *PointerReal;
```

PointerInt va conține adresa unei locații de memorie ce stochează întregi (32 biți),
PointerReal va conține adresa unei locații de memorie ce stochează double (64 biți),

> **Atenție**: tipul datelor indicate este specificat tocmai pentru că acestea sunt stocate diferit, astfel nu putem suprapune un double peste int.

> Care sunt "**ustensilele**" de care dispunem în "lupta" cu pointerii:

- **operatorul unar &** = adresa unei variabile,
- **operatorul de indirectare *** = conținutul obiectului indicat de pointer (conținutul de la adresa stocată de pointer).

Lucrul cu pointeri (continuare)

> Exemplu utilizare pointeri:

```
int x=1, y=2;  
int *ip;  
x adr.100 y adr.200 ip adr.1000  
1 2 ?*!#
```

> Efect:

- la adresa variabilei **x** (ex. 100) am pus valoarea 1,
- la adresa lui **y** (ex. 200) am pus valoarea 2,
- la adresa pointerului **ip** (ex. 1000) am pus valoarea ???

> Când un pointer este declarat, **nu conține o adresă validă**, de regulă indică către o zonă de memorie care, de cele mai multe ori nici nu aparține programului.

> După declarare, precum variabilele, **pointerul trebuie inițializat**.

Lucrul cu pointeri (continuare)

> Exemplu utilizare pointeri (continuare):

```
ip=&x;  
??? x y ip  
1 2 100  
adr.100 adr.200 adr.1000
```

> **Traducere**: la adresa de memorie a variabilei pointer **ip** se va stoca adresa (dată de operatorul &) variabilei **x**.

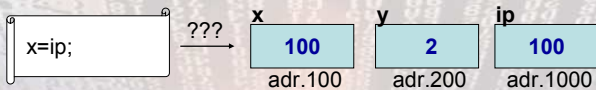
→ pointerul **ip** indică acum către variabila **x**.

```
y=*ip;  
??? x y ip  
1 1 100  
adr.100 adr.200 adr.1000
```

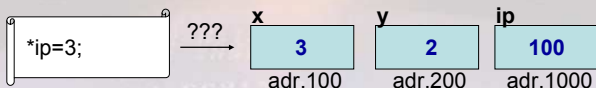
> **Traducere**: variabila **y** preia valoarea de la adresa stocată în pointerul **ip** sau cu alte cuvinte, conținutul obiectului indicat de **ip** (returnat de operatorul *).

Lucrul cu pointeri (continuare)

> Exemplu utilizare pointeri (continuare):



> *Traducere*: variabila `x` va lua valoarea variabilei pointer `ip`, și anume adresa conținută de aceasta, care este adresa lui `x`.



> *Traducere*: conținutul obiectului indicat de pointerul `ip`, și anume locația de memorie de la adresa 100 va primi valoarea 3.

→ variabila `x` are valoarea 3 (echivalent `x=3`).

Lucrul cu pointeri (continuare)

> Recapitulare:

ip înseamnă:
conținutul variabilei pointer `ip` și anume o adresă la o variabilă de tip întreg.

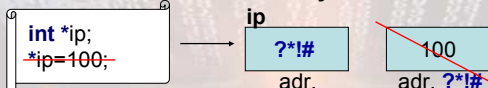
&ip înseamnă:
adresa din memorie a variabilei pointer `ip` și anume locația unde stochează adresele

***ip** înseamnă:
conținutul obiectului indicat de pointer și anume conținutul locației de memorie de la adresa memorată de pointer (adr. `ip`)

Aritmetica pointerilor

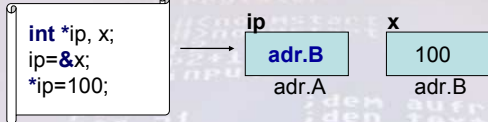
> Fiind variabile, pointerii permit efectuarea a o serie de operații specifice variabilelor.

Care este efectul acestor instrucțiuni ?



> **Greșeală tipică**: un pointer trebuie **inițializat**, deoarece altfel este posibil să indice o locație inaccesibilă.

Corect ar fi:

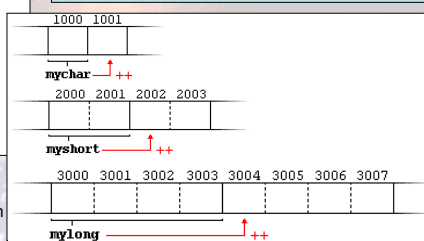


Aritmetica pointerilor (continuare)

> Exemplu 1*:

```
char *mychar;
short *myshort;
long *mylong;
mychar=1000;
myshort=2000;
mylong=3000;
mychar++;
myshort++;
mylong++;
```

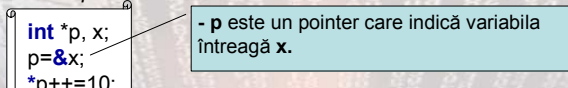
- **mychar** indică o locație de memorie pe 8 biți (1 byte),
- **myshort** indică o locație de memorie pe 16 biți (2 bytes),
- **mylong** indică o locație de memorie pe 32 biți (4 bytes),



*<http://www.cplusplus.com/doc/tutorial/pointers.html>

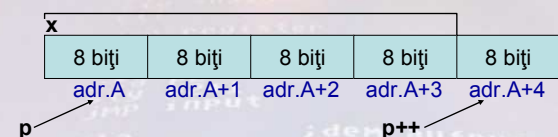
Aritmetica pointerilor (continuare)

> Exemplu 2:



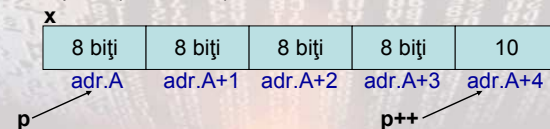
> **Atenție**: operatorii `++` și `--` au prioritate mai mare decât operatorul `*` (vezi Cursul 5).

> Astfel, `*p++` este echivalent cu `*(p++)`.



Aritmetica pointerilor (continuare)

> Exemplu 2 (continuare)



> Teoretic, `*(p++)=10` se traduce prin:

- `p` nu mai are ca valoare adresa lui `x` (`adr.A`) ci `p++`, și anume adresa `p+4bytes` (`adr.A+4`).

- la adresa stocată la `adr.A+4` se va pune valoarea 10.

> **Practic** am omis faptul că `p++` este o **incrementare post**:

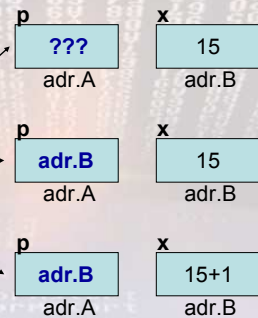
- la adresa stocată în `p` se va pune valoarea 10.

- după aceasta, `p` este incrementat și indică `adr.A+4`.

Aritmetica pointerilor (continuare)

> Exemplu 3:

```
int *p, x=15;
p=&x;
(*p)++;
```



> **(*p)++** conținutul locației de memorie indicate de **p** este mărit cu 1, echivalent cu **x++**.

Aritmetica pointerilor (continuare)

> Exemplu 4:

```
int numere[5], n;
int *p;
p=numere;
*p=10;
p++;
*p=20;
p=&numere[2];
*p=30;
p=numere + 3;
*p=40;
p=numere;
*(p+4)=50;
for (n=0; n<5; n++)
    printf("%d, ", numere[n]);
```

numere	32biți	32biți	32biți	32biți	32biți
	1000	1004	1008	1012	1016

p=1000 (p indică primul element al vectorului)

numere	32biți	32biți	32biți	32biți	32biți
	1000	1004	1008	1012	1016

p=1000+4 (adresa indicată de **p** este incrementată cu o unitate, în acest caz un int de 32 de biți)

Aritmetica pointerilor (continuare)

> Exemplu 4 (cont.)

```
int numere[5], n;
int *p;
p=numere;
*p=10;
p++;
*p=20;
p=&numere[2];
*p=30;
p=numere + 3;
*p=40;
p=numere;
*(p+4)=50;
for (n=0; n<5; n++)
    printf("%d, ", numere[n]);
```

numere	10	20	32biți	32biți	32biți
	1000	1004	1008	1012	1016

p=1004 (incrementat anterior)

p=adresa elementului de indice 2 din vector, 1008.
p indică acum spre variabila **numere[2]**.

numere	10	20	30	32biți	32biți
	1000	1004	1008	1012	1016

Aritmetica pointerilor (continuare)

> Exemplu 4 (cont.)

```
int numere[5], n;
int *p;
p=numere;
*p=10;
p++;
*p=20;
p=&numere[2];
*p=30;
p=numere + 3;
*p=40;
p=numere;
*(p+4)=50;
for (n=0; n<5; n++)
    printf("%d, ", numere[n]);
```

p va primi ca valoare adresa primului element din vectorul **numere**, 1000, la care se adaugă 3 unități de memorie, în acest caz 3*32 de biți. **p=1012**

numere	10	20	30	40	32biți
	1000	1004	1008	1012	1016

p=1000, operația + este prioritară față de *, astfel **p=1016**

numere	10	20	30	40	50
	1000	1004	1008	1012	1016

Pointeri și funcții

> În general, în limbajul C, parametrii unei funcții sunt transmiși prin **valoare**.

→ funcția apelată primește o copie a variabilei, adică valoarea variabilei copiată la o altă adresă de memorie decât cea a variabilei originale

> Exemplu:

```
void schimba(int a, int b);
```

variabilele **a** și **b** sunt transmise prin valorile acestora, deci nu vor fi modificate de funcție.

> O altă modalitate de transmitere a parametrilor este prin **referință** sau **adresă**.

→ funcția apelată primește o copie a adresei variabilei ce conține parametrul efectiv (un pointer).

Pointeri și funcții (continuare)

> Exemplu:

```
void schimba(int *pa, int *pb);
```

funcția primește două valori de adrese ce corespund locațiilor la care se află variabilele ce vrem să le modificăm.

> Detaliu funcție:

```
void schimba(int *pa, int *pb)
{
    int tmp;
    tmp=*pa;
    *pa=*pb;
    *pb=tmp;
}
```

funcția interschimbă valorile a două variabile de tip întreg.

tmp preia conținutul locației de memorie indicată de **pa** (*pa).

conținutul lui **pa** (*pa) preia conținutul lui **pb** (*pb).

conținutul lui **pb** (*pb) ia valoarea **tmp**.

Pointeri și funcții (continuare)

> Cum apelăm funcția din programul principal ?

```
int main()
{
    int a=3, b=6;
    schimba(&a, &b);
}
```

```
void schimba(int *pa, int *pb)
{
    int tmp;
    tmp=*pa;
    *pa=*pb;
    *pb=tmp;
}
```

> Să vedem ce se întâmplă în memorie ?

a	b	pa	pb
3	6	adr.a	adr.b

1.

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 24/66

Pointeri și funcții (continuare)

```
int main()
{
    int a=3, b=6;
    schimba(&a, &b);
}
```

```
void schimba(int *pa, int *pb)
{
    int tmp;
    tmp=*pa;
    *pa=*pb;
    *pb=tmp;
}
```

a	b	pa	pb	tmp
3	6	adr.a	adr.b	???

2.

a	b	pa	pb	tmp
3	6	adr.a	adr.b	3

3.

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 25/66

Pointeri și funcții (continuare)

```
int main()
{
    int a=3, b=6;
    schimba(&a, &b);
}
```

```
void schimba(int *pa, int *pb)
{
    int tmp;
    tmp=*pa;
    *pa=*pb;
    *pb=tmp;
}
```

a	b	pa	pb	tmp
6	6	adr.a	adr.b	3

4.

a	b	pa	pb	tmp
6	3	adr.a	adr.b	3

5.

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 26/66

8.2. Lucrul cu fișiere de date

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 27/66

Fișiere de date

> Datele folosite la un moment dat, în timpul execuției unui program sunt **stocate temporar** în memoria internă (ex. RAM);

> Datele ce sunt păstrate pe termen lung sunt **stocate permanent** în memoria externă (ex. HDD);

→ preluate în memoria RAM când sunt folosite;

informație brută

program: Windows 7 Explorer

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 28/66

Fișiere de date (continuare)

⇒ Pentru a putea accesa informația de pe dispozitivele externe de stocare aceasta trebuie **indexată** sau **organizată**;

A. unitățile fizice sunt împărțite (opțional) la nivel logic (nu fizic) în partiții ce sunt desemnate prin litere, ex. **C:**, **D:**, **F:** etc.

computer management Windows XP

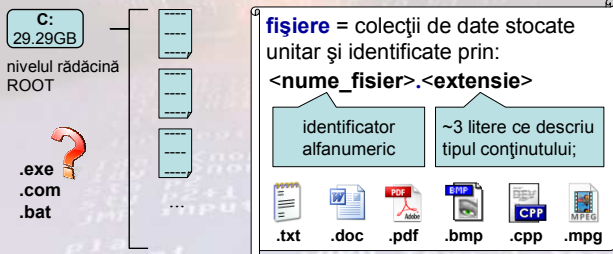
C: 29.29GB
D: 119.75GB
G: 960MB
H: 1.95GB

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU, 2016-2017 29/66

Fișiere de date (continuare)

⇒ Pentru a putea accesa informația de pe dispozitivele externe de stocare aceasta trebuie **indexată** sau **organizată**;

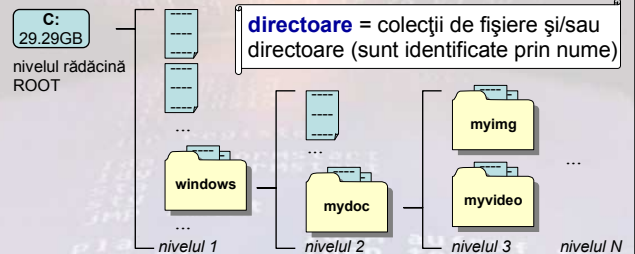
B. informația stocată pe fiecare unitate logică (partiție) este organizată ierarhic (arborescent):



Fișiere de date (continuare)

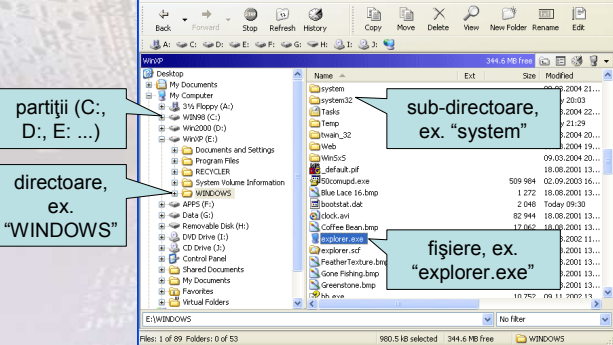
⇒ Pentru a putea accesa informația de pe dispozitivele externe de stocare aceasta trebuie **indexată** sau **organizată**;

B. informația stocată pe fiecare unitate logică (partiție) este organizată ierarhic (arborescent):



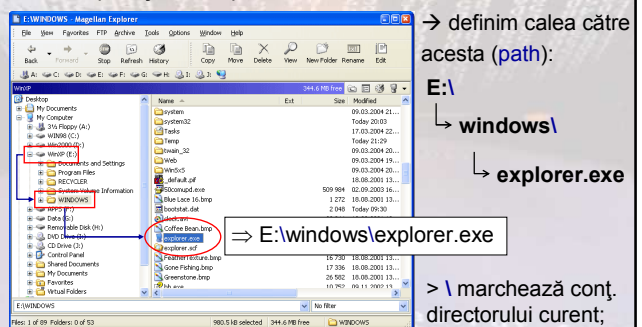
Fișiere de date (continuare)

> Exemplu:



Fișiere de date (continuare)

> Cum facem referire la un anumit fișier dintr-un anumit director, de exemplu fișierul "explorer.exe" ?



Manipularea fișierelor în C

> Citirea - scrierea datelor în fișiere reprezintă de fapt operații de **intrare** (datele sunt preluate din fișier = input) – **ieșire** (datele sunt stocate pe HDD = output);

> Uneori este avantajos să preluăm datele direct dintr-un fișier, astfel nefiind necesară intervenția utilizatorului:

- volum mare de date de intrare;
- datele de intrare sunt aceleași la fiecare rulare (ex. teste);
- datele de intrare sunt furnizate de un alt program independent de utilizator ...

> Similar, datele de ieșire ale programului pot fi stocate:

- rămân disponibile după încheierea programului;
- pot fi folosite ca date de intrare pentru alt program ...

Manipularea fișierelor în C (continuare)

> Limbajul de programare C pune la dispoziția programatorului următoarele operații de lucru cu fișiere de date:

- crearea și denumirea fișierelor;
- deschiderea unui fișier pentru manipulare date;
- citirea conținutului unui fișier (secvențial, aleator);
- scrierea datelor într-un fișier (creare fișier nou sau adăugare de date la un fișier existent);
- închiderea fișierului;

> funcțiile se găsesc în biblioteca **stdio.h**

Manipularea fişierelor în C (continuare)

> Un fişier este referenţiat printr-o variabilă pointer:

```
FILE *fisier1;  
FILE *fisier2;
```

fisier1 este un pointer la o structură de tip fişier de date,
fisier2 este un alt pointer la o structură de tip fişier de date.

> Ca şi în cazul pointerilor, fişierele trebuie iniţializate şi anume le trebuie asociat un fişier fizic de pe dispozitivul de stocare:

```
FILE *fopen(const char *nume_fisier, const char *mod);
```

returnează un pointer la fişier.

se specifică numele fişierului şi calea acestuia.

se specifică modul de deschidere al fişierului

Manipularea fişierelor în C (continuare)

```
FILE *fopen(const char *nume_fisier, const char *mod);
```

• modul de deschidere al fişierului:

- **"r"** – (read only) permite deschiderea unui fişier doar pentru a fi citit conţinutul acestuia,
- **"w"** – (write) crează un fişier gol în care se pot scrie date. Dacă fişierul cu acest nume există deja, conţinutul acestuia este şters!
- **"a"** – (append) deschide fişierul menţionat pentru a adăuga date în continuarea celor existente. Dacă fişierul nu există acesta este creat.

Manipularea fişierelor în C (continuare)

```
FILE *fopen(const char *nume_fisier, const char *mod);
```

• modul de deschidere al fişierului (continuare):

- **"r+"** – deschide un fişier pentru actualizarea datelor, permiţând atât scriere cât şi citire,
- **"w+"** – crează un fişier nou în care se vor putea scrie cât şi citi date. Dacă există un fişier cu acelaşi nume atunci conţinutul acestuia este şters.
- **"a+"** – deschide un fişier pentru adăugare de date cât şi pentru citire. Toate operaţiile de scriere vor fi efectuate exclusiv în continuarea datelor existente (protejare conţinut iniţial). Dacă fişierul nu există atunci va fi creat.

Manipularea fişierelor în C (continuare)

> **Observaţie:** modurile de deschidere enunţate anterior se referă la **fişiere de tip text** (conţinut alfanumeric), pentru fişiere binare se adaugă caracterul "b";

> Exemplul 1:

```
FILE *fisier;  
fisier=fopen("c:\\test.txt", "w");
```

De ce apare \\ ?

→ backslash este un caracter special, comanda \ + \ = "\\" !

```
if (fisier==NULL)  
printf("Fişierul nu a putut fi creat");
```

- **fişier** este un pointer la un obiect de tip FILE,

- se crează fişierul **test.txt** la locaţia specificată, şi anume pe partiţia **c:** (director rădăcină) şi se returnează controlul pentru scriere la pointerul **fişier**.

dacă pointerul este nul (nu indică nimic) → problemă la deschidere!

Manipularea fişierelor în C (continuare)

> Exemplul 2:

```
FILE *fisier;  
fisier=fopen("c:\\Demo\\Tmp\\test.txt", "w");
```

unde este creat fişierul ?

c:\Demo\Tmp\test.txt

> Exemplul 3:

```
FILE *fisier;  
fisier=fopen("test.txt", "w");
```

dar acum ?

în directorul curent

Manipularea fişierelor în C (continuare)

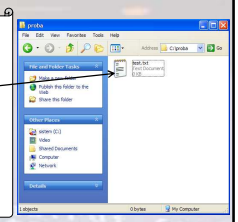
> După folosire un fişier trebuie închis (eliberare flux de date):

```
int fclose(FILE *fisier);
```

- închide fluxul de date deservit de pointerul **fişier**,
- dacă operaţia este reuşită se returnează codul 0.

> Exemplu complet:

```
FILE *fisier;  
fisier=fopen("c:\\proba\\test.txt", "w");  
if (fisier==NULL)  
printf("Fişierul nu a putut fi creat");  
else  
{ ... fclose(fisier); }
```



Manipularea fişierelor în C (continuare)

> Scrierea datelor în fişier:

```
int fprintf(FILE *fisier, const char *format, a, b, 15, ... );
```

fisier este pointerul ce referenţiază conţinutul fişierului

format (şir de caractere) specifică modul de afişare (formatare) cât şi tipul datelor afişate

după „,” se specifică variabilele sau constantele anunţate în *format*.

> Funcţia **fprintf** funcţionează identic ca funcţia **printf** doar că datele nu sunt afişate pe ecran, ci salvate într-un fişier text;
> Fişierul text poate fi văzut ca o imagine a datelor de pe ecran (acestea sunt salvate identic, păstrând modul de formatare);

Manipularea fişierelor în C (continuare)

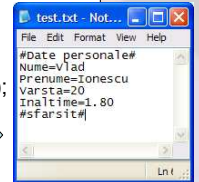
> Exemplul 1:

```
FILE *fisier;
int v=20; float i=1.8; char nume[20], prenume[20];

printf("Nume="); scanf("%s",nume);
printf("Prenume="); scanf("%s",prenume);

fisier=fopen("c:\\test.txt", "w");
fprintf(fisier, "#Date personale#\n");
fprintf(fisier, "Nume=%s\n",nume);
fprintf(fisier, "Prenume=%s\n",prenume);
fprintf(fisier, "Varsta=%d\n",v);
fprintf(fisier, "Inaltime=%.2f\n",i);
fprintf(fisier, "#sfarsit#");
fclose(fisier);
```

c:\> Nume=Vlad (enter)
Prenume=Ionescu (enter)



Manipularea fişierelor în C (continuare)

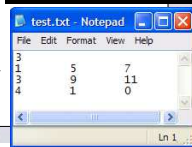
> Exemplul 2:

```
FILE *fisier;
int matrice[20][20], dim, i, j;

printf("Dim="); scanf("%d",&dim);
CitireMatrice(matrice, dim); // functie

fisier=fopen("c:\\test.txt", "w");
fprintf(fisier, "%d\n", dim);
for (i=0; i<dim; i++)
{ for (j=0; j<dim; j++)
    fprintf(fisier, "%d\t",matrice[i][j]);
  fprintf(fisier, "\n"); }
fclose(fisier);
```

c:\> Dim=3 (enter)
M[0][0]=1 (enter)
M[0][1]=5 (enter)
M[0][2]=7 (enter)
M[1][0]=3 (enter)
M[1][1]=9 (enter)
M[1][2]=11 (enter)
M[2][0]=4 (enter)
M[2][1]=1 (enter)
M[2][2]=0 (enter)



Manipularea fişierelor în C (continuare)

> Exemplul 3:

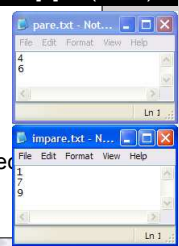
```
FILE *fpar, *fimpar;
int vect[100], dim, i;

printf("Dim="); scanf("%d",&dim);
for (i=0; i<dim; i++)
{ printf("V[%d]=",i);
  scanf("%d",&vect[i]); }

fpar=fopen("c:\\pare.txt", "w");
fimpar=fopen("c:\\impare.txt", "w");

for (i=0; i<dim; i++)
if ((vect[i] % 2)==0) fprintf(fpar,"%d\n", vect[i]);
else fprintf(fimpar, "%d\n", vect[i]);
fclose(fpar); fclose(fimpar);
```

c:\> Dim=5 (enter)
V[0]=1 (enter)
V[1]=4 (enter)
V[2]=7 (enter)
V[3]=6 (enter)
V[4]=9 (enter)



Manipularea fişierelor în C (continuare)

> Exemplul 3: continuare

```
FILE *fpar, *fimpar;
int vect[100], dim, i;

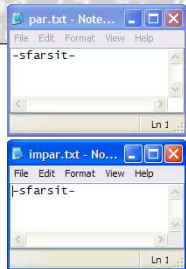
// ... citire vector
// ... scriere vector in fisier

fpar=fopen("c:\\par.txt", "a");
fimpar=fopen("c:\\impar.txt", "a");

fprintf(fpar, "-sfarsit-");
fprintf(fimpar, "-sfarsit-");

fclose(fpar);
fclose(fimpar);
```

- am deschis fişierele de date în modul de adăugare de informaţii (append) ???
- fişierele par.txt şi impar.txt nu există → vor fi create.



Manipularea fişierelor în C (continuare)

> Exemplul 3: corect

```
FILE *fpar, *fimpar;
int vect[100], dim, i;

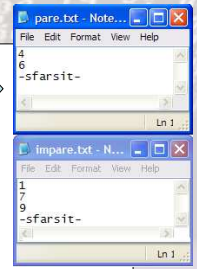
// ... citire vector
// ... scriere vector in fisier

fpar=fopen("c:\\pare.txt", "a");
fimpar=fopen("c:\\impare.txt", "a");

fprintf(fpar, "-sfarsit-");
fprintf(fimpar, "-sfarsit-");

fclose(fpar);
fclose(fimpar);
```

- fiind în modul de adăugare, textul va fi scris în continuarea datelor existente deja în fişier.



Manipularea fișierelor în C (continuare)

> Scrierea datelor în fișiere:

P

Enunț: Să se definească o structură de date ce permite stocarea următoarelor informații relative la o persoană: nume, prenume, vârstă, adresă, telefon, email. Să se realizeze o funcție ce permite citirea acestor date pentru elevii unei clase de liceu. Să se exporte datele obținute într-un fișier denumit astfel: DateEleviClasa_<numec>.txt, unde <numec> va fi specificat de utilizator.

Variabile de intrare/lucru:

```
struct elev;
struct elev clasa[40];
int NrElevi, i;
char numec[10];
```

Variabile de ieșire:

FILE *fisier;

Funcții:

```
CitireDateClasa;
ExportDateFisier;
```



Manipularea fișierelor în C (continuare)

> Programul principal:

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h> // biblioteca de lucru cu siruri de caractere

struct elev
{
    char nume[100], prenume[100], adresa[200], telefon[14], email[100];
    int varsta;
} clasa[40];

void CitireDateClasa(struct elev clasa[40], int NrElevi);
void ExportDateFisier(FILE *fisier, char numefisier[20],
                    struct elev clasa[40], char numec[10], int NrElevi);

int main()
{
    int i, NrElevi;
    char numec[10], numefisier[100];
    FILE *fisier;

    printf("Date de intrare\nNumar elevi="); scanf("%d", &NrElevi);
    printf("Numele clasei="); scanf("%s", numec); → continuare
```



Manipularea fișierelor în C (continuare)

> Programul principal (continuare):

→ continuare

```
...
int main()
{
    ...
    // citire date elevi
    CitireDateClasa(clasa, NrElevi);

    // generare nume fisier care sa contina numele clasei
    // "c:\DateEleviClasa_<numec>.txt"
    strcpy(numefisier, "c:\\DateEleviClasa_");
    strcat(numefisier, numec);
    strcat(numefisier, ".txt");

    // exportare date clasa in fisierul denumit numefisier
    ExportDateFisier(fisier, numefisier, clasa, numec, NrElevi);
}
```

- copiază șirul de caractere din dreapta în variabila șir de caractere din stânga (numefisier);

- concatenează două șiruri de caractere și le stochează în variabila din stânga (numefisier);



Manipularea fișierelor în C (continuare)

> Funcția CitireDateClasa:

```
void CitireDateClasa(struct elev clasa[40], int NrElevi)
{
    int i;

    for (i=0; i<NrElevi; i++)
    {
        printf("Elevul #%d\n", i);
        printf("Nume="); scanf("%s", clasa[i].nume);
        printf("Prenume="); scanf("%s", clasa[i].prenume);
        printf("Varsta="); scanf("%d", &clasa[i].varsta);
        printf("Adresa="); scanf("%s", clasa[i].adresa);
        printf("Telefon="); scanf("%s", clasa[i].telefon);
        printf("Email="); scanf("%s", clasa[i].email);
    }
}
```



Manipularea fișierelor în C (continuare)

> Funcția ExportDateFisier:

```
void ExportDateFisier(FILE *fisier, char numefisier[20], struct elev clasa[40],
                    char numec[10], int NrElevi)
{
    int i;
    fisier=fopen(numefisier, "w");
    if (fisier==NULL) { printf("Eroare, fisierul nu a putut fi creat!"); return; }
    else
    {
        fprintf(fisier, "#Date elevi clasa %s\n", numec);
        fprintf(fisier, "Numar total elevi %d\n", NrElevi);
        for (i=0; i<NrElevi; i++)
        {
            fprintf(fisier, "\nElevul %s %s\n", clasa[i].nume, clasa[i].prenume);
            fprintf(fisier, "varsta: %d ani\n", clasa[i].varsta);
            fprintf(fisier, "adresa: %s\n", clasa[i].adresa);
            fprintf(fisier, "telefon: %s\n", clasa[i].telefon);
            fprintf(fisier, "email: %s\n", clasa[i].email); }
        fclose(fisier);
    }
}
```



Manipularea fișierelor în C (continuare)

> Citirea datelor dintr-un fișier:

```
int fscanf(FILE *fisier, const char *format, &a, &b, ... );
```

fisier este pointerul ce referențiază conținutul fișierului

format (șir de caractere) specifică tipul datelor ce vor fi citite din fișier.

după „,” se specifică unde vor fi stocate datele citite (adrese de memorie).

> Funcția **fscanf** funcționează identic ca funcția **scanf** doar că dispozitivul de intrare nu este tastatura ci fișierul de date;

> Funcția **fscanf** returnează fie numărul de obiecte citite corect (variabile) sau în cazul în care datele nu pot fi citite returnează **EOF** (= sfârșit de fișier).



Manipularea fișierelor în C (continuare)

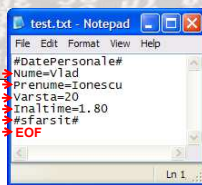
> Exemplul 1: avem la dispoziție fișierul

```
FILE *fisier;
char linie[256];
int stop;

fisier = fopen("c:\\test.txt", "r");
do
{
    stop = fscanf(fisier, "%s", linie);
    if (stop != EOF)
        printf("am citit: %s\n", linie);
} while (stop != EOF);
fclose(fisier);
```

- se deschide fișierul de date dorit în modul de citire (read);

c:\>am citit: #DatePersonale#
am citit: Nume=Vlad
am citit: Prenume=Ionescu
am citit: Varsta=20
am citit: Inaltime=1.80
am citit: #sfarsit#



Manipularea fișierelor în C (continuare)

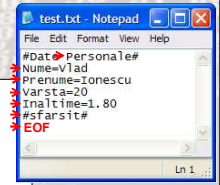
> Exemplul 2: avem la dispoziție fișierul

```
FILE *fisier;
char linie[256];
int stop;

fisier = fopen("c:\\test.txt", "r");
do
{
    stop = fscanf(fisier, "%s", linie);
    if (stop != EOF)
        printf("am citit: %s\n", linie);
} while (stop != EOF);
fclose(fisier);
```

- cum este citit textul în acest caz ? (indicație: analogie citire șiruri de caractere cu scanf)

c:\>am citit: #Date
am citit: Personale#
am citit: Nume=Vlad
am citit: Prenume=Ionescu
am citit: Varsta=20
...



Manipularea fișierelor în C (continuare)

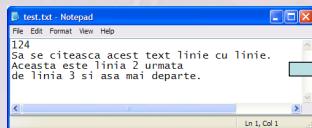
> Pentru a citi fișierul ca un șir de caractere avem nevoie de un control mai fin a ceea ce citim:

```
int fgetc(FILE *fisier);
```

fisier este pointerul ce referențiază conținutul fișierului

> Funcția **fgetc** returnează caracterul din poziția curentă la care se citește din fișier, iar după aceasta avansează automat cu un caracter (efect similar funcției **getc**).

> Avem la dispoziție fișierul text următor:



> textul de fapt arată așa:
"124\nSa se citeasca acest text linie cu linie.\nAceasta este linia 2 urmatoare de linia 3 si asa mai departe.EOF"

Manipularea fișierelor în C (continuare)

> Exemplul 3: vom citi progresiv caractere de la începutul fișierului până când întâlnim EOF:

```
FILE *fisier; char c, linie[256]; int pos=0;
fisier = fopen("c:\\test.txt", "r");

do
{
    c = fgetc(fisier);
    if ((c != '\n') && (c != EOF))
        linie[pos++] = c;
    else
    {
        linie[pos] = '\0';
        printf("%s\n", linie);
        pos = 0;
    }
} while (c != EOF);
fclose(fisier);
```

- **linie** stochează la un moment dat o linie din fișier;
- **pos** va memora poziția curentă în linie;

- citim din fișier un caracter **c** cât timp nu am ajuns la capătul fișierului;

- dacă **c** nu este sfârșit de linie și nici de fișier atunci îl adăugăm la **linie** pe **pos**;

- dacă **c** este sfârșit de linie sau de fișier atunci înseamnă că am obținut o linie completă → afișăm pe ecran;

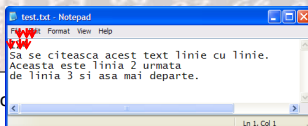
Manipularea fișierelor în C (continuare)

> Cum se execută:

```
FILE *fisier; char c, linie[256]; int pos=0;
fisier = fopen("c:\\test.txt", "r");

do
{
    c = fgetc(fisier);
    if ((c != '\n') && (c != EOF))
        linie[pos++] = c;
    else
    {
        linie[pos] = '\0';
        printf("%s\n", linie);
        pos = 0;
    }
} while (c != EOF);
fclose(fisier);
```

iterația 1: c='1', linie="1"
iterația 2: c='2', linie="12"
iterația 3: c='4', linie="124"
iterația 4: c='\n', linie="124\n"
iterația 5: c='S', linie="S"
iterația 6: c='a', linie="Sa"
iterația 7: c=' ', linie="Sa "
iterația 8: c='s', linie="Sa s"
...



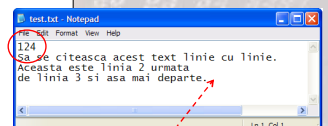
Manipularea fișierelor în C (continuare)

> După citirea linie cu linie vrem să citim numărul de la începutul fișierului, astfel:

```
FILE *fisier; char c, linie[256]; int pos=0;
fisier = fopen("c:\\test.txt", "r");

do
{
    c = fgetc(fisier);
    if ((c != '\n') && (c != EOF))
        linie[pos++] = c;
    else
    {
        linie[pos] = '\0';
        printf("%s\n", linie);
        pos = 0;
    }
} while (c != EOF);
fclose(fisier);
```

- în urma citirii pas cu pas în acest punct poziționarea în fișier este pe EOF;
→ modalitate de derulare la început a fișierului ?

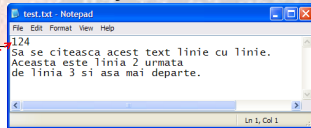


Manipularea fişierelor în C (continuare)

void rewind(FILE *fisier); — **fisier** este pointerul la fişier

> Funcţia **rewind** permite poziţionarea în fişierul curent la începutul acestuia.

```
FILE *fisier;
char c, linie[256]; int pos=0;
...
// codul anterior de citire pas cu pas
...
rewind(fisier);
fscanf(fisier, "%d", &pos);
printf("Numarul este %d", pos);
fclose(fisier);
```



c:\> 124
Sa se citeasca acest ...
... si asa mai departe.
Numarul este 124

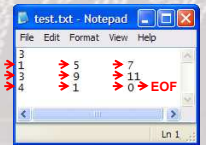
Manipularea fişierelor în C (continuare)

> Exemplul 4: se citesc valorile unei matrice pătrate de numere întregi dintr-un fişier de date. Pe prima linie este stocat numărul de elemente.

```
FILE *fisier;
int dim, matrice[100][100], i, j;
fisier = fopen("c:\\test.txt", "r");
fscanf(fisier, "%d", &dim);
for (i=0; i<dim; i++)
    for (j=0; j<dim; j++)
        fscanf(fisier, "%d", &matrice[i][j]);
fclose(fisier);
```

- se citeşte dimensiunea matricei de pe prima linie;

- se parcurg elementele matricei şi se citesc repetitiv din fişier (după fiecare citire poziţia în fişier avansează automat)



Manipularea fişierelor în C (continuare)

> Citirea/scrierea datelor în fişiere:

P **Enunţ:** Să se citească o matrice de numere reale dintr-un fişier de date de intrare. Se ştie ca pe prima linie sunt stocate numărul de linii şi numărul de coloane ale matricei. Să se calculeze valoarea maximă, valoarea minimă şi media aritmetică a valorilor. Să se adauge aceste valori la sfârşitul fişierului de date.

Variabile de intrare/lucru:

```
float matrice[100][100];
int NrL, NrC, i, j;
float max, min, medie;
FILE *fisier;
```

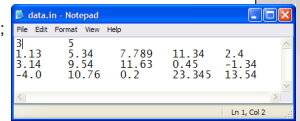
Variabile de ieşire:

```
FILE *fisier;
Funcţii:
CitireDateDinFisier;
CalculMMM;
```

Manipularea fişierelor în C (continuare)

> Funcţia CitireDateDinFisier:

```
void CitireDateDinFisier(FILE *fisier, char numef[100],
                        float matrice[100][100], int *NrL, int *NrC)
{ // functia va returna valorile matricei cat si dimensiunile acesteia prin lucrul
  // cu pointeri (variabilele sunt transmise prin adrese)
  int i, j, NrL_, NrC_;
  fisier = fopen(numef, "r");
  if (fisier == NULL) { printf("Fişierul nu a putut fi deschis!"); return; }
  else
  {
    fscanf(fisier, "%d%d", &NrL_, &NrC_);
    for (i=0; i<NrL_; i++)
      for (j=0; j<NrC_; j++)
        fscanf(fisier, "%f", &matrice[i][j]);
    *NrL = NrL_; *NrC = NrC_;
    fclose(fisier);
    printf("Au fost citite cu succes %d elemente\n", NrL_*NrC_);
  }
}
```



Manipularea fişierelor în C (continuare)

> Funcţia CalculMMM:

```
void CalculMMM(float matrice[100][100], int NrL, int NrC,
               float *max, float *min, float *medie)
{ // functia preia matrice si calculeaza valorile max, min si medie care sunt
  // returnate prin intermediul adresei (variabilele sunt transmise prin adresa)
  float max_, min_, medie_; int i, j;
  max_ = min_ = matrice[0][0];
  for (i=0; i<NrL; i++)
    for (j=0; j<NrC; j++)
    {
      if (matrice[i][j] > max_) max_ = matrice[i][j];
      if (matrice[i][j] < min_) min_ = matrice[i][j];
      medie_ += matrice[i][j];
    }
  if (NrC*NrL==0) medie_ = 0;
  else medie_ /= NrC*NrL;
  *max = max_; *min = min_; *medie = medie_;
}
```

Manipularea fişierelor în C (continuare)

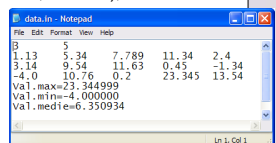
> Programul principal:

```
#include <stdio.h>
#include <stdlib.h>

int main ()
{
  FILE *fisier;
  int NrL, NrC, i, j;
  float matrice[100][100], max, min, medie;

  CitireDateDinFisier(fisier, "c:\\data.in", matrice, &NrL, &NrC);
  CalculMMM(matrice, NrL, NrC, &max, &min, &medie);

  fisier = fopen("c:\\data.in", "a");
  fprintf(fisier, "Val.max=%f\n", max);
  fprintf(fisier, "Val.min=%f\n", min);
  fprintf(fisier, "Val.medie=%f\n", medie);
  fclose(fisier);
}
```



Sfârșitul Cursului 8