Tehnici de programare - TP

Cursul 5 – Fișiere text. Fișiere binare

Ș.l. dr. ing. Cătălin Iapă

catalin.iapa@cs.upt.ro

Structuri. typedef

Fișiere text

Fișiere binare

Să ne amintim: Structuri

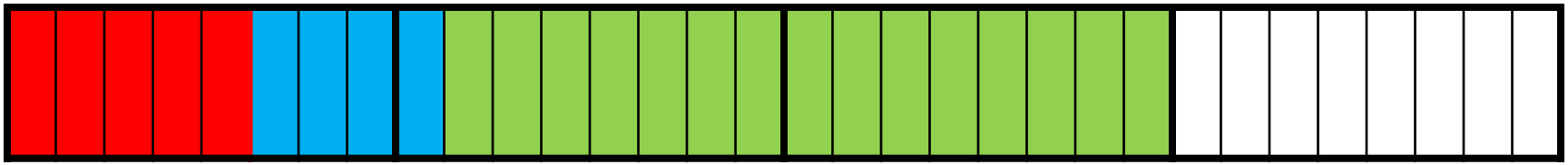
```
struct Produs{
   char nume[200];
   float pret;
   int stoc;
};
```

```
struct Produs p1, p2={“ciocolata“, 12, 100}, v[10], *p;
p1=p2;
p2.pret = 7.5;

v[3].pret=98;
strcpy(v[3].nume, ”paine”);

p = (struct Produs*) malloc (1 * sizeof(struct Elem));
p->n=3;
```

Structuri cu câmpuri pe biți

```
struct DataBiti {
    unsigned int zi:5;
    unsigned int luna:4;
    int an:15;
    };
```

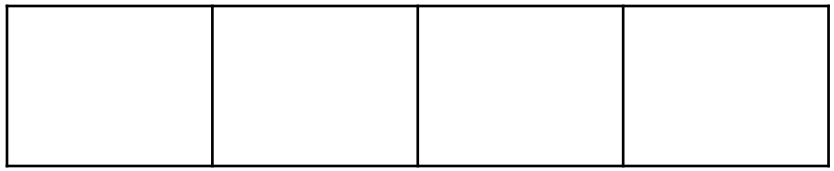
Dimensiune: 24 biți (se mărește la multiplu de 32 biți, 4 bytes, 1 int)



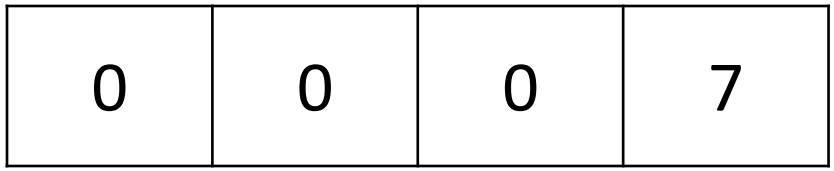
zi luna an spațiu nefolosit

Uniuni

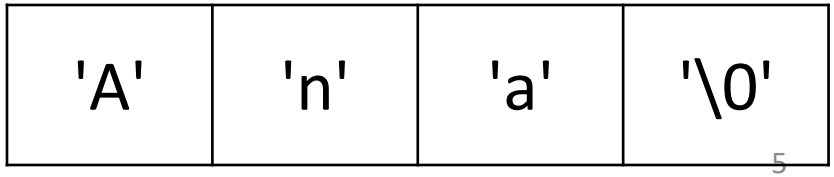
```
union int_sau_string {
        int a;
        char s[4];
} x;
```



```
x.a=7;
```




```
strcpy(x.s, "Ana");
```



typedef

În general typedef se folosește în două situații:

- Când avem de-a face cu tipuri mai complexe, pentru a nu trebui de fiecare dată să scriem tot tipul
- Când dorim să descriem mai bine intenția cu care folosim un anumit tip de date

typedef asociază un nume unei definiții de tip:

typedef tip nume;

După acest typedef, nume va putea fi folosit pentru a substitui tipul specificat.

typedef – exemple de folosire

*typedef char * String;*

String s1;

După ce s-a folosit typedef pentru a se defini String ca fiind tipul char*, s-a putut folosi String peste tot în program în locul lui char*.

typedef unsigned int uint;

uint a,b;

Aceasta definește un nou tip de date numit uint care este un alias pentru tipul de date unsigned int. Astfel, în loc să folosim unsigned int de fiecare dată când avem nevoie de acest tip de date, putem folosi uint.

typedef la structuri

Putem folosi *typedef la structuri*, pentru a da un nou nume (un nume mai scurt) tipului definit de ele.

Când definim o structură de date, numele acesteia nu este obligatoriu, ci poate să lipsească, având astfel structuri anonime.

struct {int x,y;} pt1,pt2;

typedef la structuri

Structură normală

```
struct Data{
    int zi;
    int luna;
    int an;
    };

struct Data d1,d2;
int a = sizeof(struct Data);
```

Structură definită cu typedef

```
typedef struct {
    int zi;
    int luna;
    int an;
    } Data;

Data d1,d2;
int a = sizeof (Data);
```

Structuri. typedef

Fișiere text

Fișiere binare

Interacțiuni ale programului C

Programul scris în C poate interacționa și în exteriorul acestuia astfel:

- Preia informații de la tastatură
- Afișează informații pe ecran
- Primește informații de la sistemul de operare prin argumente în linia de comandă
- Citește sau afișează date din fișiere text
- Citește sau afișează date din fișiere binare

Fișiere text

Fișierele care conțin informații sub formă de text în diferite formate, care pot fi vizualizate și modificate cu orice editor text și pot fi citite și înțelese direct de către om se numesc **fișiere text**.

Spre exemplu codul sursă în limbajul C este păstrat în fișiere text. Formatele (.txt, .xml, .html, .json, .csv) sunt exemple de fișiere text.

Fișierele sunt stocate de regulă pe disc (hard-disk, stick USB, CD, DVD, etc.).

Deschiderea și închiderea unui fișier

Pentru a se putea lucra cu fișiere, acestea trebuie mai întâi deschise cu fopen()

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

- *nume_fisier* - numele fișierului care va fi deschis sau întreaga cale unde este reținut în calculator
- *mod* - modul în care va fi deschis fișierul. Este un șir din una sau mai multe litere, având semnificațiile de mai jos:
  - **r** - (read) fișierul este deschis pentru citire (se va permite doar citirea datelor din el), începând cu începutul fișierului.
  - **w** - (write) fișierul este deschis pentru scriere. Dacă nu există, se va crea un fișier nou. Dacă fișierul există deja, atunci i se șterge tot conținutul anterior.
  - **a** - (append) fișierul este deschis pentru adăugare (scriere la sfârșit)

Deschiderea și închiderea unui fișier

Dacă *fopen* reușește să deschidă fișierul, ea va returna un pointer către un descriptor (handler) de fișier. Acest pointer are tipul **FILE*** și prin intermediul lui se vor realiza toate operațiile cu fișierul. Dacă *fopen* nu reușește să deschidă fișierul, va returna NULL.

Întotdeauna se va testa dacă deschiderea unui fișier a avut loc cu succes.

Deschiderea și închiderea unui fișier

După ce s-au terminat operațiile cu un fișier, acesta trebuie închis cu funcția:

int ***fclose****(FILE *fis)*

Dacă *fis* este NULL sau este un fișier deja închis, *fclose* nu are niciun efect.

Deschiderea și închiderea unui fișier

```
FILE *f;

int main() {
f = fopen("fis.txt", "w");
if( f == NULL) exit(1);

//prelucrarea fisierului

fclose(f);
return 0;
}
```

Scrierea într-un fișier text

Pentru scriere vom deschide fișierul în mod de scriere (**w** - write), sau de adăugare (**a** - append) iar apoi vom folosi funcții cum sunt **fprintf**, **fputs**, **fputc**:

```
FILE *f;
f = fopen("fis.txt", "w");
if( f == NULL) exit(1);

char sir[] = "Exemplu", c = 'A';
fprintf(f, "%s",sir);
fputs(sir, f);
fputc(c,f);

fclose(f);
```

Citirea din fișier text

Dacă se specifică în avans câte date vom avea de citit (de exemplu citind prima linie din fișier), vom citi datele din fișier folosind o buclă *for*.

Dacă nu știm de la început câte date sunt și dorim să citim tot fișierul, va trebui să avem o condiție de testare a sfârșitului de fișier sau a faptului că nu se mai pot citi date.

Dacă folosim pentru citire funcția *fscanf*, valoarea returnată de ea (de tip *int*) are următoarea semnificație:

- dacă a apărut o eroare de citire din fișier, se returnează EOF
- în caz de succes, *fscanf* returnează numărul de valori citite.
- dacă acest număr este 0, înseamnă că s-a ajuns la sfârșit de fișier.

Citirea din fișier text

```
FILE *g;
g = fopen("fis.txt", "r");
if( g == NULL) exit(1);

char sir[21] , c;
fscanf(g, "%20s", sir);          // fscanf citește cuvinte – citește text până când întâlnește un spațiu, un tab, un enter sau EOF

fgets(sir, 20, g);                    // fgets citește fraze – citește până când întâlnește enter sau EOF (sfârșit de fișier)
c = fgetc(g);

fclose(f);
```

Citirea din fișier text – feof(f)

Putem verifica dacă s-a ajuns la sfârșitul fișierului cu ajutorul funcției *feof(f)*.

Funcția *feof(f)* returnează true (diferit de 0), doar dacă s-a ajuns la sfârșit de fișier.

Funcția *feof* se poate folosi și la citirea din fișier, pentru a se testa dacă s-a ajuns la sfârșit (ex: *while(!feof(fis)){...citire..}*), dar se poate folosi pentru aceasta și direct valoarea returnată de funcțiile de citire.

Structuri. typedef

Fișiere text

Fișiere binare

Fișiere binare

Fișierele binare sunt destinate folosirii lor direct de către diverse aplicații. În general ele păstrează informația în același fel în care ea este păstrată în memoria calculatorului.

Aplicațiile pot folosi mult mai ușor fișierele binare decât pe cele text, deoarece nu mai trebuie:

- să convertească datele din binar la text (ex: *“fprintf(fis,”%f”,x);”*) înainte de a le scrie într-un fișier text sau
- să le convertească din text în binar, la citirea dintr-un fișier text (ex: *“fscanf(fis,”%f”,&x);”*)

Citirea din fișier text – feof(f)

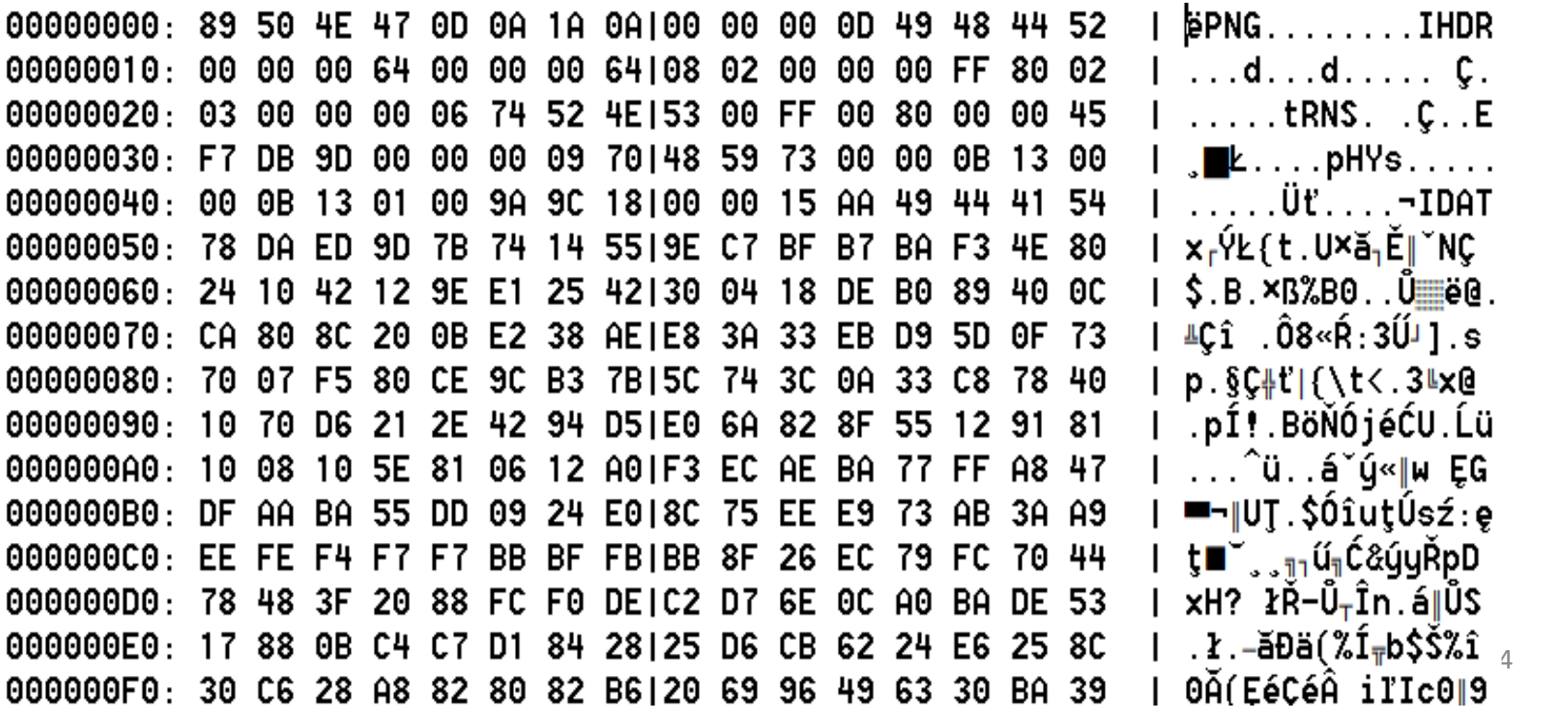
Exemple de fișiere binare sunt:

- arhive (zip, gz, 7z, rar),
- audio-video (mp3, mp4, avi, mpeg),
- imagini (jpg, png, gif),
- executabile sau cod (exe, dll, so, jar).
- inclusiv unele formate destinate documentelor (ex: epub, doc (Microsoft Word)) sunt tot fișiere binare,
- iar altele (ex: pdf) sunt o combinație de fișiere text cu unele secțiuni binare.

Fișiere binare

Pentru a vizualiza conținutul unui fișier binar putem folosi:

- în Linux: hexdump -C nume_fisier
- în Windows: Total Commander, se selectează fișierul de vizualizat, se dă comanda View (tasta F3) și din meniul Options se selectează Hex (tasta 3)



Deschiderea și închiderea fișierelor binare

În C, fișierele binare se deschid și se închid la fel ca cele text, folosind *fopen/fclose*, folosind "wb" sau "rb" la *fopen* (write binary, read binary).

```
FILE *f;
f=fopen("1.dat","wb")
if(f == NULL){
        printf("nu se poate deschide fisierul\n");
        exit(EXIT_FAILURE);
        }
 // scrierea in fisier
fclose(fis);
```

Scrierea într-un fișier binar

Pentru a scrie într-un fișier binar se folosește funcția **fwrite** declarată în *stdio.h*:

*size_t **fwrite**(void *elemente, size_t dim_element, size_t nr_elemente, FILE *fisier);*

Funcția *fwrite* are următorii parametri:

- **elemente** - un pointer la zona de memorie ce conține elementele care vor fi scrise în fișier
- **dim_element** - dimensiunea în octeți a unui element care va fi scris în fișier. Se poate afla cu *sizeof*
- **nr_elemente** - numărul de elemente de scris
- **fisier** - fisierul destinație

```
int n = 108, v[]={1, 2, 3, 4, 5};
fwrite(&n,sizeof(int),1,fis);
fwrite(v,sizeof(int),5,fis);
```

Scrierea într-un fișier binar

```
typedef struct{
   char nume[28];
   float pret;
   }Produs;

Produs produse[1000];
int nProduse;

fwrite(produse,sizeof(Produs),nProduse,fis);
```

//pentru scrierea în fișier a întregii baze de date am avut nevoie de o singură instrucțiune *fwrite*. Dacă am fi folosit un fișier text, ar fi fost nevoie să iterăm toate produsele din baza de date. Acesta este un exemplu de ce aplicațiile pot folosi mai ușor și mai eficient formate binare decât formate text.

Scrierea într-un fișier binar

Dacă adăugăm în baza de date 2 produse (mere:5, paine:3), fișierul va avea un conținut de forma:

```
00000000: 6D 65 72 65 00 00 CC CC|CC CC CC CC CC CC CC CC  | mere..╠╠╠╠╠╠╠╠
00000010: CC CC CC CC CC CC CC CC|CC CC CC CC 00 00 A0 40  | ╠╠╠╠╠╠╠╠╠╠╠╠..á@
00000020: 70 61 69 6E 65 00 00 CC|CC CC CC CC CC CC CC CC  | paine..╠╠╠╠╠╠╠╠
00000030: CC CC CC CC CC CC CC CC|CC CC CC CC 00 00 40 40  | ╠╠╠╠╠╠╠╠╠╠╠╠..@@
```

```
typedef struct{
    char nume[28];
    float pret;
    }Produs;
```

Citirea dintr-un fișier binar

Pentru citire din fișiere binare se folosește funcția **fread**, care exact aceleași argumente și valoare returnată ca și funcția *fwrite*:

*size_t **fread**(void *elemente, size_t dim_element, size_t nr_elemente, FILE *fisier);*

Funcția *fread* are următorii parametri:

- **elemente** - un pointer la zona de memorie în care se vor citi elementele din fișier
- **dim_element** - dimensiunea în octeți a unui element care va fi citit din fișier. Se poate afla cu *sizeof*
- **nr_elemente** - numărul de elemente de citit
- **fisier** - fisierul sursă

Funcția returnează numărul de elemente citite integral din fișier.

Accesul la fișiere

```
//exemplu de program care copiază conținutul unui fișier în altul

#include <stdio.h>
int main() {
   FILE *in_file, *out_file;
   char buffer[1000];
   int bytes_read;
   in_file = fopen("input.bin", "rb");
   out_file = fopen("output.bin", "wb");
   while ((bytes_read = fread(buffer, 1, 1000, in_file)) > 0) {
      fwrite(buffer, 1, bytes_read, out_file);
   }
   fclose(in_file);
   fclose(out_file);
   return 0;
}
```

Accesul la fișiere

Odată fişierul creat, limbajul de programare C permite două *modalităţi de acces* la componentele sale:

- **acces secvenţial** : componentele sunt prelucrate strict în ordinea în care sunt înregistrate în fişier.

- **acces direct** : componentele se pot prelucra în orice ordine, specificând poziţia în fişier a componentei ce urmează să fie prelucrată.

Accesul la fișiere

Pentru accces direct oriunde în fișier, nu doar secvențial vom folosi funcția fseek:

*int **fseek**(FILE *fis,long offset,int reper)*

-setează poziția curentă din fișier, folosind ca punct de referință reperul specificat.

Reperul poate fi: **SEEK_SET** - începutul fișierului, **SEEK_CUR** - poziția curentă, **SEEK_END** - sfârșitul fișierului.

fseek(fis,0,SEEK_END) - mută poziția curentă la sfârșit de fișier.

fseek(fis,-5,SEEK_CUR) - mută poziția crt. cu 5 octeți în stânga.

*long **ftell**(FILE *fis)*

- returnează poziția curentă (offsetul) din fișier, față de începutul acestuia. La această poziție vor avea loc următoarele operații de citire sau de scriere.

Vă mulțumesc!