PROGRAMAREA CALCULATOARELOR

Andrei Patrascu

andrei.patrascu@fmi.unibuc.ro

Secția Calculatoare si Tehnologia Informatiei, anul I, 2018-2019

Cursul 10

PROGRAMA CURSULUI

Introducere

- Algoritmi
- Limbaje de programare.
- Introducere în limbajul C. Structura unui program C.

Fundamentele limbajului C

- Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
- Tipuri derivate de date: tablouri, șiruri de caractere, structuri, uniuni, câmpuri de biți, enumerări, pointeri
- Instrucțiuni de control
- Directive de preprocesare. Macrodefiniții.
- Funcții de citire/scriere.
- Etapele realizării unui program C.

Fișiere text

- Funcții specifice de manipulare.

Funcții (1)

- Declarare și definire. Apel. Metode de trasmitere a paramerilor. Pointeri la funcții.

Tablouri și pointeri

- Legătura dintre tablouri și pointeri
- Aritmetica pointerilor
- Alocarea dinamică a memoriei
- Clase de memorare

Șiruri de caractere

- Funcții specifice de manipulare.

Fișiere binare

- Funcții specifice de manipulare.

Structuri de date complexe și autoreferite

- Definire și utilizare

Funcții (2)

- Funcții cu număr variabil de argumente.
- Preluarea argumentelor funcției main din linia de comandă.
- Programare generică.

CURSUL DE AZI

1. Functii pentru manipulare blocuri de memorie
2. Fișiere binare: funcții specifice de manipulare
3. Structuri autoreferite

FUNCȚII PREDEFINITE PENTRU MANIPULAREA BLOCURILOR DE MEMORIE

- copierea elementelor unui tablou a într-un alt tablou b:
  - nu se poate face prin atribuire (b=a), întrucât a și b sunt pointeri constanți;
  - copierea se face element cu element folosind instrucțiuni repetitive (for, while);
  - pentru stringuri (tablouri de caractere) avem funcțiile predefinite strcpy și strncpy:
    - char* strcpy(char *d, char* s);
      - copiază șirul sursă s în șirul destinație d;
      - returnează adresa șirului destinație
      - șirul rezultat are un ‘\0’ la final
    - char* strncpy(char *d, char* s, int n);
      - copiază primele n caractere șirul sursă s în șirul destinație d;
      - returnează adresa șirului destinație
      - șirul rezultatul NU are un ‘\0’ la final

FUNCȚII PREDEFINITE PENTRU MANIPULAREA BLOCURILOR DE MEMORIE

- copierea elementelor unui tablou a într-un alt tablou b:
  - nu se poate face prin atribuire (b=a), întrucât a și b sunt pointeri constanți;
  - copierea se face element cu element folosind instrucțiuni repetitive (for, while);
  - pe cazul general (a și b nu sunt neaparat tablouri de caractere) putem folosi funcții pentru manipularea blocurilor de memorie: memcpy, memmove;
    - lucrează la nivel de octet fără semn (unsigned char)
    - alte funcții pentru manipularea blocurilor de memorie: memcmp, memset, memchr

FUNCȚII PREDEFINITE PENTRU MANIPULAREA BLOCURILOR DE MEMORIE

- copierea unui tablou – funcțiile **memcpy** și **memmove**
  - antet: void* memcpy(void *d, const void* s, int n);
    - copiază primii n octeți din sursa s în destinația d;
    - returnează un pointer la începutul zonei de memorie destinație d;
    - un fel de strcpy extins (merge și pe alte tipuri de date, nu numai pe char-uri)
    - nu se oprește la octeți = 0 (funcția strcpy se oprește la octeți ce au valoarea 0 = sfârșit de string);
  - presupune că șirurile destinație și sursa nu se suprapun
    - dacă cele două șiruri se suprapun funcția prezintă undefined behaviour (comportament nedefinit)

FUNCȚII PREDEFINITE PENTRU MANIPULAREA BLOCURILOR DE MEMORIE

- copierea unui tablou – funcțiile **memcpy** și **memmove**
  - antet: void* memcpy(void *d, const void* s, int n);

exempluMemcpy.c

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

int main()
{
    int a[] = {25,-36, 0, 91, 7415};
    int *b= (int*) malloc(sizeof(a));
    memcpy(b,a,sizeof(a));
    int i;
    for (i=0;i<sizeof(a)/sizeof(int);i++)
        printf("%d ",b[i]);
    printf("\n");

    float a1[] = {2.0, 3.5, -1.2};
    float b1[3];
    memcpy(b1,a1,sizeof(a1));
    for (i=0;i<sizeof(a1)/sizeof(float);i++)
        printf("%f ",b1[i]);
    printf("\n");

    char c[50]="Ana are mere";
    memcpy(c+8,c,12); puts(c);

    return 0;
}
```

```
25 -36 0 91 7415
2.000000 3.500000 -1.200000
Ana are Ana are mere
```

FUNCȚII PREDEFINITE PENTRU MANIPULAREA BLOCURILOR DE MEMORIE

- copierea unui tablou – funcțiile **memcpy** și **memmove**
  - antet: void* memmove(void *d, const void* s, int n);
    - copiază primii n octeți din sursa s în destinația d;
    - returnează un pointer la începutul zonei de memorie destinație d;
    - identică cu funcția memcpy + tratează cazurile de suprapunere dintre d și s
  - nu contează că șirurile destinație d și sursă s se suprapun
    - folosește un buffer intern pentru copiere

FUNCȚII PREDEFINITE PENTRU MANIPULAREA BLOCURILOR DE MEMORIE

- copierea unui tablou – funcțiile **memcpy** și **memmove**
  - antet: void* memmove(void *d, const void* s, int n);

exempluMemmove.C

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

int main ()
{
    char t[] = "memmove este foarte folositor.............";
    memmove(t + 20, t + 13, 16);
    puts(t);
    return 0;
}
```

```
memmove este foarte foarte folositor.....
```

FUNCȚII PREDEFINITE PENTRU MANIPULAREA BLOCURILOR DE MEMORIE

- funcție care elimină toate aparițiile unui șir t într-un șir s

```
int main()
{
    char s[100],t[100];
    strcpy(s,"abbbccca");
    strcpy(t,"bc");
    eliminaAparitii(s,t);
    printf("%s\n",s);
    return 0;
}
```

Trebuie sa obtin
“abbcca”

FUNCȚII PREDEFINITE PENTRU MANIPULAREA BLOCURILOR DE MEMORIE

- funcție care elimină toate aparițiile unui șir t într-un șir s

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

void eliminaAparitii(char *s, char* t)
{
    char *p = strstr(s,t);
    while(p != NULL)
    {
        memmove(p,p+strlen(t),s + strlen(s)- (p + strlen(t)) + 1);
        p = strstr(s,t);
    }
}

int main()
{
    char s[100],t[100];
    strcpy(s,"abbbccca");
    strcpy(t,"bc");
    eliminaAparitii(s,t);
    printf("%s\n",s);
    return 0;
}
```

Nu obțin ceea ce trebuie, unde am greșit?

FUNCȚII PREDEFINITE PENTRU MANIPULAREA BLOCURILOR DE MEMORIE

- funcție care elimină toate aparițiile unui șir t într-un șir s

exempluMemmove2.c

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

void eliminaAparitii(char *s, char* t)
{
    char *p = strstr(s,t);
    while(p != NULL)
    {
        memmove(p,p+strlen(t),s + strlen(s)- (p + strlen(t)) + 1);
        p = strstr(p,t);
    }
}

int main()
{
    char s[100],t[100];
    strcpy(s,"abbbccca");
    strcpy(t,"bc");
    eliminaAparitii(s,t);
    printf("%s\n",s);
    return 0;
}
```

abbcca

Daca pun p = strstr(p+1,t) ce se intampla?

MANIPULAREA BLOCURILOR DE MEMORIE

- setarea unor octeți la o valoare – funcția **memset**
  - antet: void* memset(void *d, char val, int n);
  - în zona de memorie dată de pointerul d, sunt setate primele n poziţii (octeți) la valoarea dată de val. Funcţia returnează şirul d.

exempluMemset.c

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main ()
5  {
6      char t[] = "nu prea vrem sa vina vacanta!!!";
7      memset(t, '-', 8);
8      puts(t);
9      return 0;
10 }
```

```
--------vrem sa vina vacanta!!!
```

MANIPULAREA BLOCURILOR DE MEMORIE

- căutarea unui octet într-un tablou – funcția **memchr**
  - antet: void* memchr(const void *d, char c, int n);
  - detemină prima apariție a octetului c în zona de memorie dată de pointerul d și care conține n octeți. Funcţia returnează pointerul la prima apariție a lui c în d sau NULL, dacă c nu se găsește în d.

exempluMemchr.c

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main ()
5  {
6      char t[] = "nu prea vrem sa vina vacanta!!!";
7      char *p = memchr(t,'m', 25);
8      puts(p);
9
10     return 0;
11 }
```

```
m sa vina vacanta!!!
```

MANIPULAREA BLOCURILOR DE MEMORIE

- compararea a două tablouri pe octeți – funcția **memcmp**
  - antet: int memcmp(const void *s1, const void * s2, int n);
  - compară primii **n** octeți corespondenți începând de la adresele **s1** și **s2**.
  - returnează 0 dacă octeții sunt identici, ceva mai mic decât 0 dacă **s1** $<_L$ **s2**, ceva mai mic decât 0 dacă **s1** $>_L$ **s2**

MANIPULAREA BLOCURILOR DE MEMORIE

- compararea a două tablouri pe octeți – funcția **memcmp**
  - antet: int memcmp(const void *s1, const void * s2, int n);
  - compară primii **n** octeți corespondenți începând de la adresele **s1** și **s2**.

exempluMemcmp.c

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

int main ()
{
    char t[] = "Ana are mere!!!";
    char s[] = "Ana are pere!!!";
    int i = memcmp(t,s,6);
    printf("%d \n",i);
    i = memcmp(t,s,strlen(t));
    printf("%d \n",i);

    int v[] = {1,2,4,5,6};
    int w[] = {1,2,3,7,8};
    i = memcmp(v,w,8);
    printf("%d \n",i);
    i = memcmp(v,w,sizeof(v));
    printf("%d \n",i);

    return 0;
}
```

```
0
-3
0
1
```

FIȘIERE BINARE

- **fișier binar = șir de octeți neformatat pe linii care este stocat pe suport magnetic/optic. Octeții nu sunt considerați ca fiind coduri de caractere.**

- un fişier binar este format în general din articole de lungime fixă, fără separatori între articole. Un articol poate conţine:
  - un singur octet
  - un număr binar (pe 2, 4 sau 8 octeţi)
  - structură cu date de diferite tipuri

FIȘIERE BINARE

- **fișier binar = șir de octeți neformatat pe linii care este stocat pe suport magnetic/optic. Octeții nu sunt considerați ca fiind coduri de caractere.**

- Diferenta fisiere text – fisiere binare:

- un fişier binar se accesează ca o succesiune de octeţi, cărora funcţiile de citire şi scriere din fişier nu le dau nici o interpretare.

- un fişier text se accesează ca o succesiune de linii de text de lungime variabilă (încheiate cu un terminator de linie : ‘\n’) utilizând un set dedicat de funcţii din biblioteca standard.

FIȘIERE BINARE

- **fișier binar = șir de octeți neformatat pe linii care este stocat pe suport magnetic/optic. Octeții nu sunt considerați ca fiind coduri de caractere.**

- **FILE *fopen(char *nume_fisier, char *mod_deschidere)**
  - nume_fisier = numele fisierului
  - mod_deschidere = șir de caracter ce precizează tipul de acces la fișier:

Mod	Semnificaţie
r	Deschide un fişier tip text pentru a fi citit
w	Creează un fişier tip text pentru a fi scris
a	Adaugă într-un fişier tip text
rb	Deschide un fişier de tip binar pentru a fi citit
wb	Creează un fişier de tip binar pentru a fi scris
ab	Adaugă într-un fişier de tip binar
r+	Deschide un fişier tip text pentru a fi citit/scris
w+	Creează un fişier tip text pentru a fi citit/scris
a+	Adaugă în sau creează un fişier tip text pentru a fi citit/scris
r+b	Deschide un text în binar pentru a fi citit/scris
w+b	Creează un fişier de tip binar pentru a fi citit/scris
a+b	Adaugă sau creează un fişier de tip binar pentru a fi citit/scris

FUNCȚII DE CITIRE/SCRIERE

int fwrite(void *tablou, int dim_element, int nr_elem, FILE *f)

- scrie în fișierul referit de f cel mult nr_elem elemente de dimensiune dim_element de la adresa tablou;

int fread(void *tablou, int dim_element, int nr_elem, FILE *f

- citește cel mult nr_elem elemente de dimensiune dim_element din fisierul referit de f la adresa tablou.

FUNCȚII DE CITIRE/SCRIERE

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

int main()
{
    FILE *f, *g;
    int x = 1936880995;

    f = fopen("fisier_nrb.out","wb");
    g = fopen("fisier_nrt.out","w");

    fwrite(&x, sizeof(int), 1, f);
    fprintf(g, "%d", x);

    printf("Cod ASCII c = %d\n", 'c');
    printf("Cod ASCII u = %d\n", 'u');
    printf("Cod ASCII r = %d\n", 'r');
    printf("Cod ASCII s = %d\n", 's');
    //printf("%d\n", (99<<24) + (117<<16) + (114<<8)+ 115);
    printf("%d\n", (115<<24) + (114<<16) + (117<<8)+ 99);


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

FUNCȚII DE CITIRE/SCRIERE

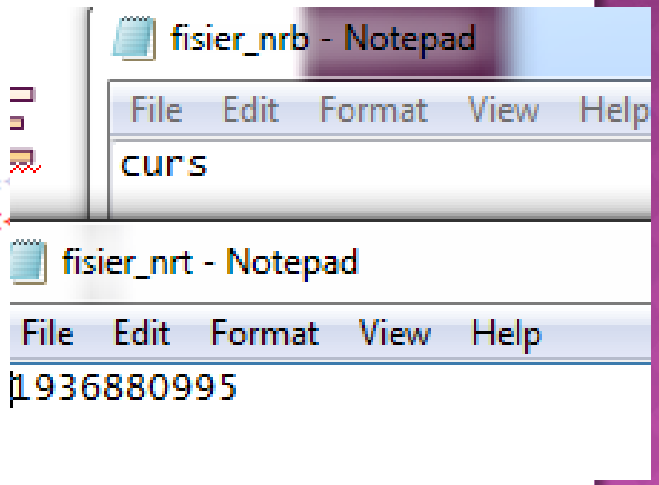
```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    FILE *f, *g;
    int x = 1936880995;

    f = fopen("fisier_nrb.
    g = fopen("fisier_nrt.

    fwrite(&x, sizeof(int), 1, f);
    fprintf(g, "%d", x);

    printf("Cod ASCII c = %d\n", 'c');
    printf("Cod ASCII u = %d\n", 'u');
    printf("Cod ASCII r = %d\n", 'r');
    printf("Cod ASCII s = %d\n", 's');
    //printf("%d\n", (99<<24) + (117<<16) + (114<<8)+ 115);
    printf("%d\n", (115<<24) + (114<<16) + (117<<8)+ 99);


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

```
Cod ASCII c = 99
Cod ASCII u = 117
Cod ASCII r = 114
Cod ASCII s = 115
1936880995

Process returned 0 (0x0)   execution time :
```

FUNCȚII DE CITIRE/SCRIERE

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

int main()
{
    FILE *f, *g;
    int x = 1936880995;

    f = fopen("fisier_nrb.out","wb");
    g = fopen("fisier_nrt.out","w");

    fwrite(&x, sizeof(int), 1, f);
    fprintf(g, "%d", x);

    printf("Cod ASCII c = %d\n", 'c');
    printf("Cod ASCII u = %d\n", 'u');
    printf("Cod ASCII r = %d\n", 'r');
    printf("Cod ASCII s = %d\n", 's');
    //printf("%d\n", (99<<24) + (117<<16) + (114
    printf("%d\n", (115<<24) + (114<<16) + (117<

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

```
Cod ASCII c = 99
Cod ASCII u = 117
Cod ASCII r = 114
Cod ASCII s = 115
1936880995

Process returned 0 (0x0)   execution time :
```



FUNCȚII DE CITIRE/SCRIERE

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

int main()
{
    FILE *f;
    int x; char c;
    f = fopen("fisier_nrb.out","rb");

    if (f == NULL)
    {printf("Deschidere esuata!"); exit(1);}

    while (fread(&x, sizeof(int), 1, f)==1)
        printf("x = %d\n",x);
    fclose(f);

    f = fopen("fisier_nrb.out","rb");

    while (fread(&c, sizeof(char), 1, f)==1)
        printf("c = %d\n",c);
```

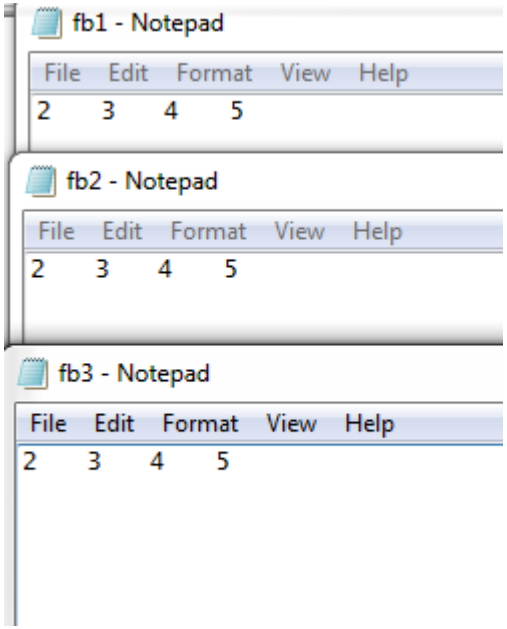
FUNCȚII DE CITIRE/SCRIERE

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

int main()
{
    FILE *f;
    int x; char c;
    f = fopen("fisier_nrb.(

    if (f == NULL)
    {printf("Deschidere esuata!"); exit(1);}

    while (fread(&x, sizeof(int), 1, f)==1)
        printf("x = %d\n",x);
    fclose(f);

    f = fopen("fisier_nrb.out","rb");

    while (fread(&c, sizeof(char), 1, f)==1)
        printf("c = %d\n",c);
```

```
x = 1936880995
c = 99
c = 117
c = 114
c = 115

Process returned 0 (0x0)   execution t
Press any key to continue.
```

FUNCȚII DE CITIRE/SCRIERE

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

int main()
{
    FILE *f1, *f2, *f3;
   f1 = fopen("fb1.txt","wb");
   f2 = fopen("fb2.txt","wb");
   f3 = fopen("fb3.txt","wb");

    if ((f1 == NULL)||(f2 == NULL)||(f3 == NULL))
    {printf("Deschidere esuata!"); exit(1);}

    int v[4] = {50,51,52,53},i;

    for (i=0;i<4;i++)
        fwrite(&v[i],sizeof(int),1,f1);

    fwrite(v,sizeof(int),4,f2);

    fwrite(v,4*sizeof(int),1,f3);

    fclose(f1);  fclose(f2); fclose(f3);

    return 0;
}
```



FUNCȚII DE CITIRE/SCRIERE

```
void afisare(int* v, int dim)
{
int i;
for (i = 0; i<dim; i++)
    printf("%d ",v[i]);
printf("\n");
}

int main()
{
    FILE *f1, *f2, *f3;
   f1 = fopen("fb1.txt","rb");
   f2 = fopen("fb2.txt","rb");
   f3 = fopen("fb3.txt","rb");

    if ((f1 == NULL)||(f2 == NULL)||(f3 == NULL))
    {printf("Deschidere esuata!"); exit(1);}

    int v[5],i;

    for (i=0;i<4;i++)
        fread(&v[i],sizeof(int),1,f1);

    afisare(v,4);

    fread(v,sizeof(int),4,f2);
    afisare(v,4);

    fread(v,4*sizeof(int),1,f3);
    afisare(v,4);
```

FUNCȚII DE CITIRE/SCRIERE

```
void afisare(int* v, int dim)
{
int i;
for (i = 0; i<dim; i++)
    printf("%d ",v[i]);
printf("\n");
}

int main()
{
    FILE *f1, *f2, *f3;
   f1 = fopen("fb1.txt","rb");
   f2 = fopen("fb2.txt","rb");
   f3 = fopen("fb3.txt","rb");

    if ((f1 == NULL)||(f2 == NULL)||(f3 == NULL))
    {printf("Deschidere esuata!"); exit(1);}

    int v[5],i;

    for (i=0;i<4;i++)
        fread(&v[i],sizeof(int

    afisare(v,4);

    fread(v,sizeof(int),4,f2);
    afisare(v,4);

    fread(v,4*sizeof(int),1,f3
    afisare(v,4);
```

```
50 51 52 53
50 51 52 53
50 51 52 53

Process returned 0 (0x0)   execution t
Press any key to continue.
```

FUNCȚII DE CITIRE/SCRIERE

- Scrierea unei structuri

exempluFisierBinar5.c

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

typedef struct
{
    int varsta;
    char nume[20];
} student;

int main()
{
    FILE *f;
    student st;
    int i;
    f = fopen("/Users/bogdan/FMI/PP/Bogdan/2016_2017/SubiecteExamen/date.in","wb");
    if (f==NULL)
        {
            printf("Fisierul date.in nu se poate crea \n");
            exit(0);
        }
    printf("Nume student = ");scanf("%s",&st.nume);
    printf("Varsta student = ");scanf("%d",&st.varsta);
    fwrite(&st,sizeof(st),1,f);

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

FUNCȚII DE CITIRE/SCRIERE

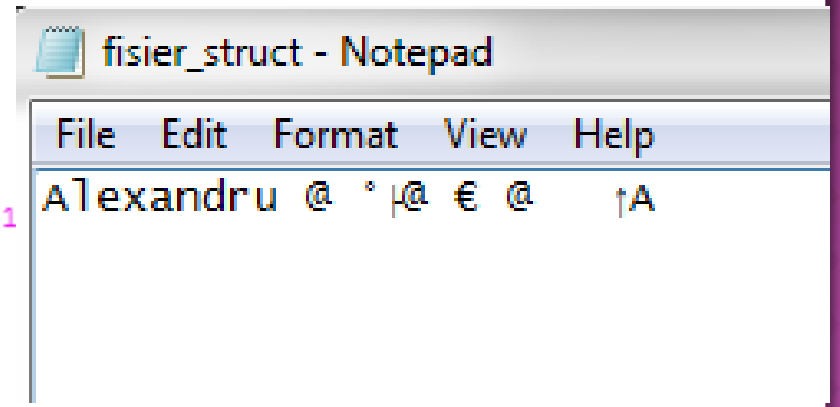
- Scrierea unei structuri

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct
    {
        char nume[20];
        float medie;
    } student;


int main()
{
    FILE *f;
    f = fopen("fisier_struct.out","wb");

    if (f == NULL)
    {printf("Deschidere esuata!"); exit(1);}

    student s;
    strcpy(s.nume,"Alexandru");
    s.medie = 9.5;
    fwrite(&s, sizeof(s), 1, f);

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

FUNCȚII DE CITIRE/SCRIERE

- Scrierea unei structuri

```
#include <stdio.h>
#include <stdlib.h>
typedef struct
    {
        char nume[20];
        float medie;
    } student;


int main()
{
    FILE *f;
    f = fopen("fisier_struct.out","wb");

    if (f == NULL)
    {printf("Deschidere esuata!"); exit(1

    student s;
    strcpy(s.nume,"Alexandru");
    s.medie = 9.5;
    fwrite(&s, sizeof(s), 1, f);

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



FUNCȚII DE CITIRE/SCRIERE

- Citirea unei structuri

```c
#include <stdlib.h>
typedef struct
    {
        char nume[20];
        float medie;
    } student;


int main()
{
    FILE *f;
    f = fopen("fisier_struct.out","rb");

    if (f == NULL)
    {printf("Deschidere esuata!"); exit(1);}

    student s;
    fread(&s, sizeof(s), 1, f);
    printf("Student cu numele: %s si media %f \n",s.nume,s.medie);

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

FUNCȚII DE CITIRE/SCRIERE

- Citirea unei structuri

```c
#include <stdlib.h>
typedef struct
    {
        char nume[20];
        float medie;
    } student;


int main()
{
    FILE *f;
    f = fopen("fisier_struct.out","rb");

    if (f == NULL)
    {printf("Deschidere esuata!"); exit(1);}

    student s;
    fread(&s, sizeof(s), 1, f);
    printf("Student cu numele: %s si media %f \n",s.nume,s.medie);

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

```
Student cu numele: Alexandru si media 9.500000

Process returned 0 (0x0)   execution time : 0.008 s
Press any key to continue.
```

FUNCȚII DE POZIȚIONARE ÎNTR-UN FIȘIER

- în C ne putem poziționa pe un anumit octet din fișier.

  Funcțiile care permit poziționarea (cele mai importante) sunt:

- **long int ftell(FILE *f)**
  - întoarce numărul octetului curent față de începutul fișierului;
  - (dimensiunea maximă a unui fișier în C este de 2^{31}-1 octeți ~ 2GB)
- **int fseek(FILE *f, int nr_octeti, int origine)**
  - mută pointerul de fișier f pe octetul numărul nr_octeti in raport cu origine
  - origine – 3 valori posibile:
    - SEEK_SET (= 0) - început de fișier
    - SEEK_CUR (=1) – poziția curentă
    - SEEK_END (=2) – sfârșit de fișier

ALTE FUNCȚII PENTRU LUCRUL CU FIȘIERELE

- **void rewind(FILE *f)**
  - repoziționarea pointerului f asociat fișierului la începutul său.
- **int remove(char * nume_fisier);**
  - șterge fișierul cu numele = nume_fișier. Întoarce 0 în caz de succes, 1 în caz de eroare;
- **int rename(char *nume_vechi,char *nume_nou);**
  - redenumește fișierul cu numele = nume_vechi cu nume_nou. Întoarce 0 în caz de succes, 1 în caz de eroare;
- **char *tmpnam(char* nume_fisier)**
  - furnizează un nume de fisier pe care îl pune în nume_fisier care nu există în directorul curent

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;
}
```

COMPARAȚIE PERFORMANȚĂ FGETC, FGETS, FREAD

Studiu preluat de la: 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