



Programare procedurala

- suport de curs -

Dobrovat Anca - Madalina

An universitar 2016 – 2017

Semestrul I

Curs 10

Evaluarea activităților didactice pe semestrul 1

- ❑ studenții au posibilitatea să evalueze activitățile cadrelor didactice
- ❑ evaluare = completarea unui chestionar pentru fiecare disciplină (curs/seminar/laborator) ce conține 10 întrebări = 3-5 minute
- ❑ evaluările au caracter anonim, fiecare student se va loga folosind un token nenominal de unică folosință primit de la secretariat de către șeful de grupă și distribuit apoi membrilor grupei.
- ❑ evaluarea se desfășoară în perioada 5 - 15 ianuarie
- ❑ concluziile din evaluarea de anul trecut vor fi făcute publice într-un raport (cel mai probabil în acest weekend)

Evaluarea activităților didactice pe semestrul 1

- ❑ studenții au posibilitatea să evalueze activitățile cadrelor didactice
- ❑ evaluare = completarea unui chestionar pentru fiecare disciplină (curs/seminar/laborator) ce conține 10 întrebări = 3-5 minute
- ❑ evaluările au caracter anonim, fiecare student se va loga folosind un token nenominal de unică folosință primit de la secretariat de către șeful de grupă și distribuit apoi membrilor grupei.
- ❑ evaluarea se desfășoară în perioada 5 - 15 ianuarie
- ❑ concluziile din evaluarea de anul trecut vor fi făcute publice într-un raport (cel mai probabil în acest weekend)



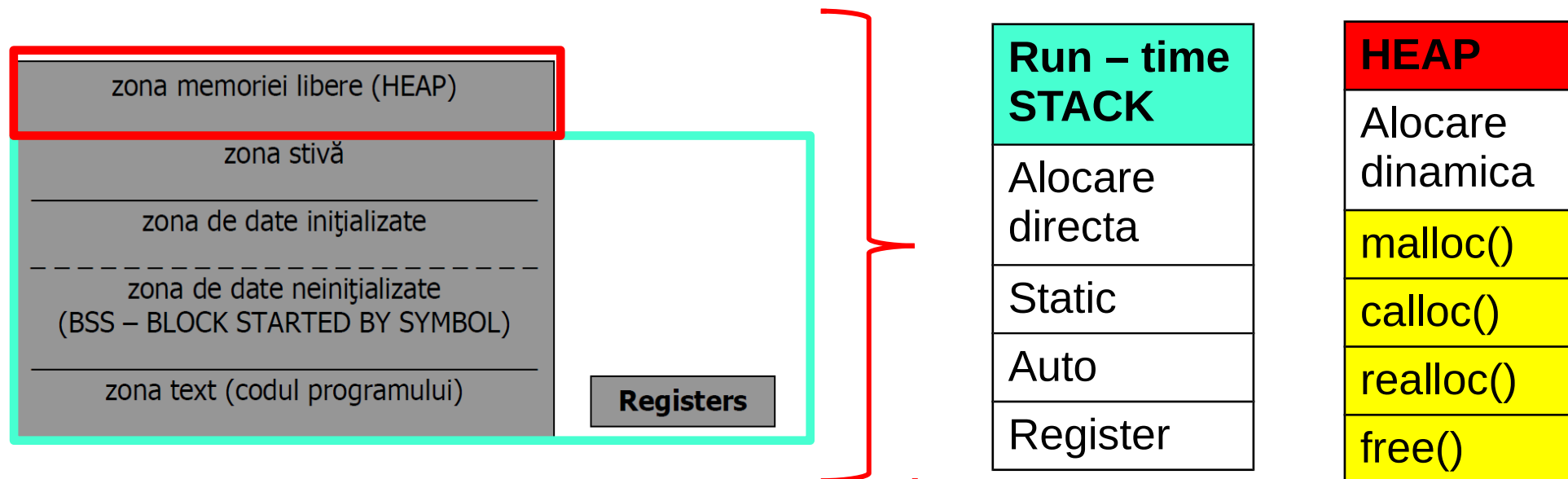
Agenda cursului

0. Alocarea dinamica a memoriei (recapitulare)
1. Funcții predefinite pentru manipularea blocurilor de memorie
 - memcpy, memmove, memchr, memcmp, etc
2. Clase de alocare / memorare
3. Subprograme recursive
4. Fisiere text si fisiere binare



0. Alocarea dinamica a memoriei (recapitulare)

Harta simplificată a memoriei la rularea unui program [1]



heap-ul este o zonă predefinită de memorie (de dimensiuni foarte mari) care poate fi accesată de program pentru a stoca date și variabile

[1] <http://www.utgjiu.ro/ing/down/poo-capitolul04.pdf>



0. Alocarea dinamica a memoriei (recapitulare)

Avantaje ale alocarii dinamice

- memoria necesara este alocata (si / sau eliberata) in timpul executiei programului (cand e nevoie) si nu la compilarea programului
- un bloc de memorie alocat dinamic poate fi redimensionat dupa necesitati.

Dezavantaje

- mai mult de codat – alocarea memoriei trebuie facuta explicit in cod
- posibile bug-uri

Functia malloc()

Returneaza adresa de inceput a unui bloc de memorie alocat in HEAP (daca exista suficient spațiu liber).

Pointerul în care păstrăm adresa returnată de malloc va fi plasat în zona de memorie statică.



0. Alocarea dinamica a memoriei (recapitulare)

Funcția calloc()

Este echivalenta cu funcția malloc(), dar, pe langa alocare de memorie pentru un bloc, realizeaza si inițializarea zonei alocate

Funcția realloc() - Redimensionarea blocurilor alocate dinamic

Primește ca parametri adresa de memorie a unui bloc deja alocat și noua dimensiune si returneaza noua lui adresa de memorie (daca exista suficient spatiu pentru realocare) sau NULL.

In caz de succes → blocul poate să fie mutat la o nouă locație de memorie, dar tot conținutul va fi păstrat.

Funcția free()

Elibereaza zona de memorie alocata in decursul executarii programului.



0. Alocarea dinamica a memoriei (recapitulare)

Functia malloc()

Expl. Citirea si afisarea unui sir de numere reale

```
float *a;  
int n, i;  
scanf("%d", &n);  
a = (float *) malloc(n * sizeof(float));  
if (!a) printf("Eroare alocare.\n");  
  
for (i = 0; i < n; i++) {  
    printf("a[%d] = ", i);  
    scanf("%f", &a[i]);  
}  
  
printf("\n\n\n");  
for (i = 0; i < n; i++)  
    printf("%.2f ", a[i]);  
printf("\n");  
  
free(a);
```

Citire lungime sir si alocarea
unui numar EXACT de octeti

Citirea elementelor sirului

Afisarea elementelor sirului cu doua
zecimale exacte

Eliberarea memoriei ocupate in HEAP



0. Alocarea dinamica a memoriei (recapitulare)

Greseli frecvente

1. Zone marcate de SO ca fiind ocupate, dar inutilizabile, intrucat se pierde adresa de inceput a blocului.

```
int *v, *temp;  
v = (int*) malloc(10*sizeof(int));  
temp = (int*) malloc(10*sizeof(int));  
  
printf("Adresa lui v[0] = %p\nAdresa lui temp[0] = %p\n\n",v,temp);  
temp = v;  
printf("Adresa lui v[0] = %p\nAdresa lui temp[0] = %p\n\n",v,temp);
```

```
C:\Users\Ank\Desktop\curs10\bin\Debug\curs10.exe  
Adresa lui v[0] = 003D0FC8  
Adresa lui temp[0] = 003D0FF8  
  
Adresa lui v[0] = 003D0FC8  
Adresa lui temp[0] = 003D0FC8
```



0. Alocarea dinamica a memoriei (recapitulare)

Greseli frecvente

2. Alocarea de memorie pentru o variabila locala – la iesirea din functie, variabila se sterge, dar zona ramane alocata si inutilizabila

```
void f()
{
    int *v = (int*) malloc(10*sizeof(int));
    free(v);
}

int main()
{
    f();
    return 0;
}
```



0. Alocarea dinamica a memoriei (recapitulare)

Greseli frecvente

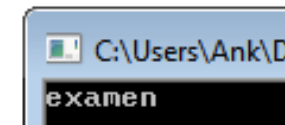
3. Pointeri fara zona de memorie alocata

Incorect

```
char s[20] = "examen";  
char *t;  
strcpy(t,s);  
puts(t);
```

Corect

```
char s[20] = "examen";  
char *t;  
  
t = (char *) malloc(strlen(s)+1);  
strcpy(t,s);  
puts(t);  
  
return 0;
```





Agenda cursului

0. Alocarea dinamica a memoriei (recapitulare)
1. Funcții predefinite pentru manipularea blocurilor de memorie
 - memcpy, memmove, memchr, memcmp, etc
2. Clase de alocare / memorare
3. Subprograme recursive
4. Fisiere text si fisiere binare



1. Funcții predefinite pentru manipularea blocurilor de memorie

Fie 2 tablouri a si b. **Copierea elementelor lui a in b:**

- nu se poate face prin atribuire ($b=a$), întrucât a și b sunt pointeri constanți, ci element cu element folosind instrucțiuni repetitive;
- daca s si b sunt stringuri (tablouri de caractere) avem funcțiile predefinite: **strcpy** și **strncpy**;
- pe caz general (a și b nu sunt neaparat tablouri de caractere) : functii pentru manipularea blocurilor de memorie: **memcpy, memmove**:
 - x) lucrează la nivel de octet fără semn (unsigned char)
 - x) alte funcții pentru manipularea blocurilor de memorie: **memcmp, memset, memchr**



1. Funcții predefinite pentru manipularea blocurilor de memorie

Funcția **memcpy**

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;
- echivalentul general pentru strcpy, dar, spre deosebire de aceasta, nu se oprește la octeți = 0;
- **presupune că șirurile destinație și sursa nu se suprapun, în caz contrar, pe unele compilatoare, funcția prezintă undefined behaviour (comportament nedefinit)**



1. Funcții predefinite pentru manipularea blocurilor de memorie

Funcția **memcpy**

```
int main()
{
    float a[ ] = {-2.3, 4.99, 45.77, -23.11};
    float *b;
    int i,n;

    b = (float *) malloc(sizeof(a));
    memcpy(b,a,sizeof(a));

    n = sizeof(a)/sizeof(float);

    for (i = 0; i<n; i++)
        printf("%f ",b[i]);

    printf("\n\n");
}
```

```
-2.300000 4.990000 45.770000 -23.110001
```

```
char c[50] = "Undefined behaviour";

memcpy(c+10, c, strlen(c));
puts(c);
```

```
Undefined Undefined behaviour
```

Pe unele compilatoare e posibil
sa obtinem altceva decat "Undefined
Undefined behaviour"



1. Funcții predefinite pentru manipularea blocurilor de memorie

Funcția **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 (folosește un buffer intern pentru copiere).

```
char c[50] = "Test cu memmove";  
  
memmove(c+10, c, strlen(c));  
puts(c);
```

```
Test cu meTest cu memmove
```




1. Funcții predefinite pentru manipularea blocurilor de memorie

Funcția **memset**

- setarea unor octeți la o valoare

antet: **void* memset(void *d, const int val, int n);**

-în zona de memorie dată de pointerul d, sunt setate primele n poziții (octeți) la valoarea dată de val. Funcția returnează șirul d.

- **Valoarea e transmisa ca un int, dar functia umple blocurile de memorie folosind conversia catre unsigned char.**



1. Funcții predefinite pentru manipularea blocurilor de memorie

Funcția **memset**

```
char d[20] = "Test cu memset";  
memset(d, '@', 7);  
puts(d);  
  
printf("\n\n");  
  
int e[3] = {-25, 10, 65};  
memset(e, '$', 8);  
int i;  
for(i=0; i<3; i++)  
    printf("%c ", e[i]); // afisarea conversiei catre char  
printf("\n\n");
```

Terminal output showing the result of the memset function. The first line displays "Test cu memset" where the first 7 characters have been replaced by '@'. The second line shows the conversion of the integers -25, 10, and 65 to their corresponding ASCII characters '\$', '\$', and 'A'.



1. Funcții predefinite pentru manipularea blocurilor de memorie

Funcția **memchr**

- cautarea unui octet într-un tablou

antet: **void* memchr(const void *d, char c, int n);**

- determină prima apariție a octetului c în zona de memorie dată de pointerul d și care conține n octeți.
- returnează pointerul la prima apariție a lui c în d sau NULL, dacă c nu se găsește în d.

```
char f[20] = "Test cu memchr";  
  
char *g = memchr(f, 'e', strlen(f));  
puts(g);
```

```
est cu memchr
```



1. Funcții predefinite pentru manipularea blocurilor de memorie

Funcția **memcmp**

- compararea a doua tablouri

antet: **void* memcmp(const void *s1, const void *s2, int n);**

- compara primii n octeți corespondenți începând de la adresele s1 și s2.
- returnează 0 dacă octeții sunt identici, ceva mai mic decât 0 dacă $s1 < s2$, ceva mai mic decât 0 dacă $s1 > s2$



1. Funcții predefinite pentru manipularea blocurilor de memorie

Funcția **memcmp**

```
char s1[ ] = "Sir de comparat 1";  
char s2[ ] = "Sir pentru comparat 2";  
int x,y;  
x = memcmp(s1,s2,4);  
y = memcmp(s1,s2,sizeof(s1));  
printf("x = %d, y = %d\n\n",x,y);
```

```
int s3[ ] = {10, 23, 44, -567, -99};  
int s4[ ] = {10, 23, 44, 5, 99};  
int z,t;  
z = memcmp(s3,s4,12);  
t = memcmp(s3,s4,sizeof(s3));  
printf("z = %d, t = %d\n\n",z,t);
```

```
x = 0, y = -1  
z = 0, t = 1
```



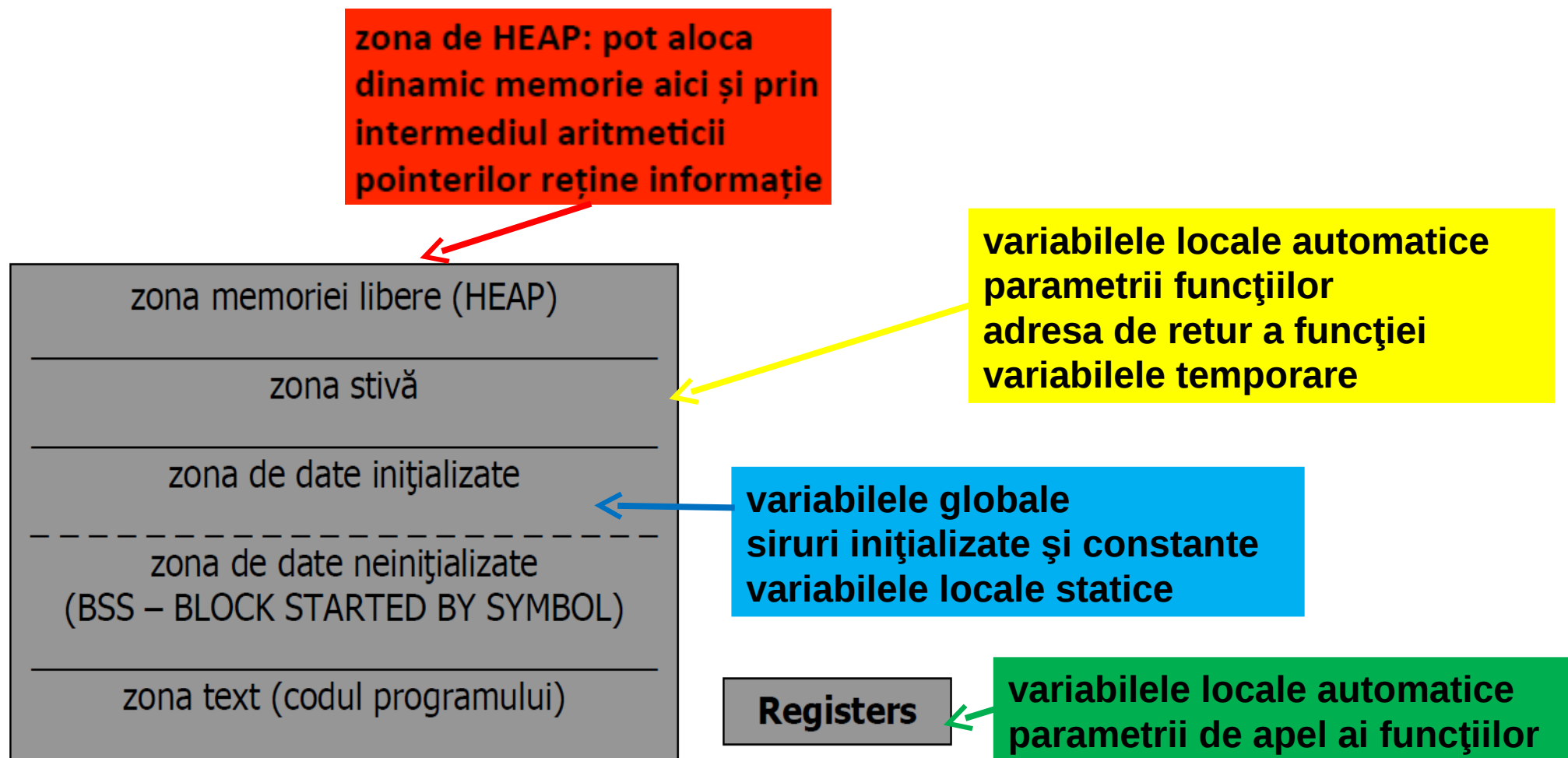
Agenda cursului

0. Alocarea dinamica a memoriei (recapitulare)
1. Funcții predefinite pentru manipularea blocurilor de memorie
 - memcpy, memmove, memchr, memcmp, etc
2. Clase de alocare / memorare
3. Subprograme recursive
4. Fisiere text si fisiere binare



2. Clase de alocare / memorare

Harta simplificata a memoriei la incarcarea programului ^[1]



[1] <http://www.utgjiu.ro/ing/down/poo-capitolul04.pdf>



2. Clase de alocare / memorare

Clasa de alocare a unei variabile defineste urmatoarele caracteristici

- locatia de memorie unde se rezerva spatiu pentru variabila,
- durata de viata,
- vizibilitatea,
- modalitatea de initializare.

Clase de alocare: **auto(matic)**

register

static (intern)

static extern



2. Clase de alocare / memorare

Clasa de alocare: **auto**

- este implicita (variabile locale);
- se specifica prin cuvantul cheie **auto**;
- in mod implicit toate variabilele locale sunt memorate pe stiva (in memoria volatila);
- spatiul de memorie se alocă la executie;
- variabilele automate sunt vizibile numai în corpul funcțiilor/ instrucțiunilor compuse în care au fost declarate;



2. Clase de alocare / memorare

Clasa de alocare: **auto**

- variabilele automate sunt vizibile numai în corpul funcțiilor/instrucțiunilor compuse în care au fost declarate; la revenirea din execuția funcțiilor/instrucțiunilor compuse variabilele se elimină și stiva revine la starea dinaintea apelului;

-nu sunt inițializate;

-parametrii funcțiilor sunt transmisi, de asemenea, prin stiva (**de la dreapta la stânga!**).

```
void f(int *x, int *y)
{
    auto int t;
    t = *x;
    *x = *y;
    *y = t;
}
```



2. Clase de alocare / memorare

Clasa de alocare: **register**

- se specifica prin cuvantul cheie **register**;
- asemenea celor din clasa auto **diferenta**: daca e posibil, ele vor fi memorate in registrele UC si nu in RAM (se cere un acces rapid (registrul procesorului) la o variabilă. Nu se garantează că cererea va fi satisfăcută);
- ***Nu exista adresa de memorie asociata.**
- nr limitat de variabile in registri (compilerul le trece in clasa auto, dar cu pastrarea restrictiei *);



2. Clase de alocare / memorare

Clasa de alocare: **register**

- numai parametri și variabilele automate de tipul **int**, **char** și **pointer** pot fi declarate ca variabile registru;
- nu se pot manipula tablouri de registrii (ar fi nevoie de dereferențiere de adresa elementului de început al tabloului, ori variabilele de tip register nu au adresa de memorie asociata);
- parametrii formali pot fi declarati in clasa register

Expl:

```
register int a;
```

```
void f(register int x);
```



2. Clase de alocare / memorare

Clasele de alocare: **auto si register**

Concluzii:

Variabilele din clasele auto si register nu isi pastreaza valoarea de la un apel altul al functiei in care sunt definite.

Compilatoarele moderne nu au nevoie de asemenea declarații, ele reușesc să optimize codul mai bine decât am putea noi prin declararea variabilelor de tip register

Mai multe detalii: Albeanu G – Programare procedurala (note de curs 2013)



2. Clase de alocare / memorare

Clasa de alocare: **static intern**

- este specificata prin cuvantul cheie **static**;
- sunt memorate in locatii fixe de memorie (au permanent asociata aceeasi adresa);
- durata de viata - pe parcursul executarii intregului program.
- variabilele globale sunt implicit din clasa static

O variabila din clasa static intern (**obligatoriu trebuie initializata**) se initializeaza numai la primul apel (prin codul generat!)



2. Clase de alocare / memorare

Clasa de alocare: **static intern**

```
void afis (int x)
{
    static int y = 25;
    printf("Exemplificare modificare variabila y: %d\n\n", y);
    y++;
}

int main(void) {
    afis (1);
    afis (1);
    afis (1);

    return 0;
}
```

C:\Users\Ank\Desktop\Lab6\bin\Debug\Lab6.exe

```
Exemplificare modificare variabila y: 25
Exemplificare modificare variabila y: 26
Exemplificare modificare variabila y: 27
Process returned 0 (0x0)   execution time
Press any key to continue.
```

Se initializeaza la primul apel al functiei, iar la celelalte apeluri isi schimba doar valoarea. Static = adresa fixa!



2. Clase de alocare / memorare

Clasa de alocare: **static intern**

- variabilele din clasa static nu sunt globale, sunt vizibile numai în funcțiile/ blocurile de funcții în care au fost declarate

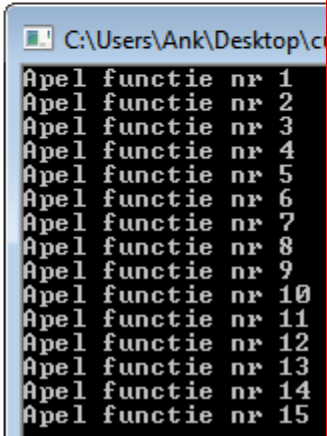
-Aplicatie – recursivitate

Se numara cate
apeluri generează o
funcție

```
int f (int x)
{
    static int nr_apeluri = 0;
    nr_apeluri++;
    printf("Apel functie nr %d \n", nr_apeluri);

    if (x <= 1) return 1;
    return f(x-1) + f(x-2);
}

int main()
{
    int x = f(5);
    return 0;
}
```





2. Clase de alocare / memorare

Clasa de alocare: **static extern**

- se specifica prin cuvantul cheie **extern**;
- implica accesarea unei varibile definita la nivel exterior (alt modul (fișier) decat cel curent)

Accesarea functiilor definite in alt modul (de exemplu functii de biblioteca!):

- prototipul in modulul in care este folosita functia

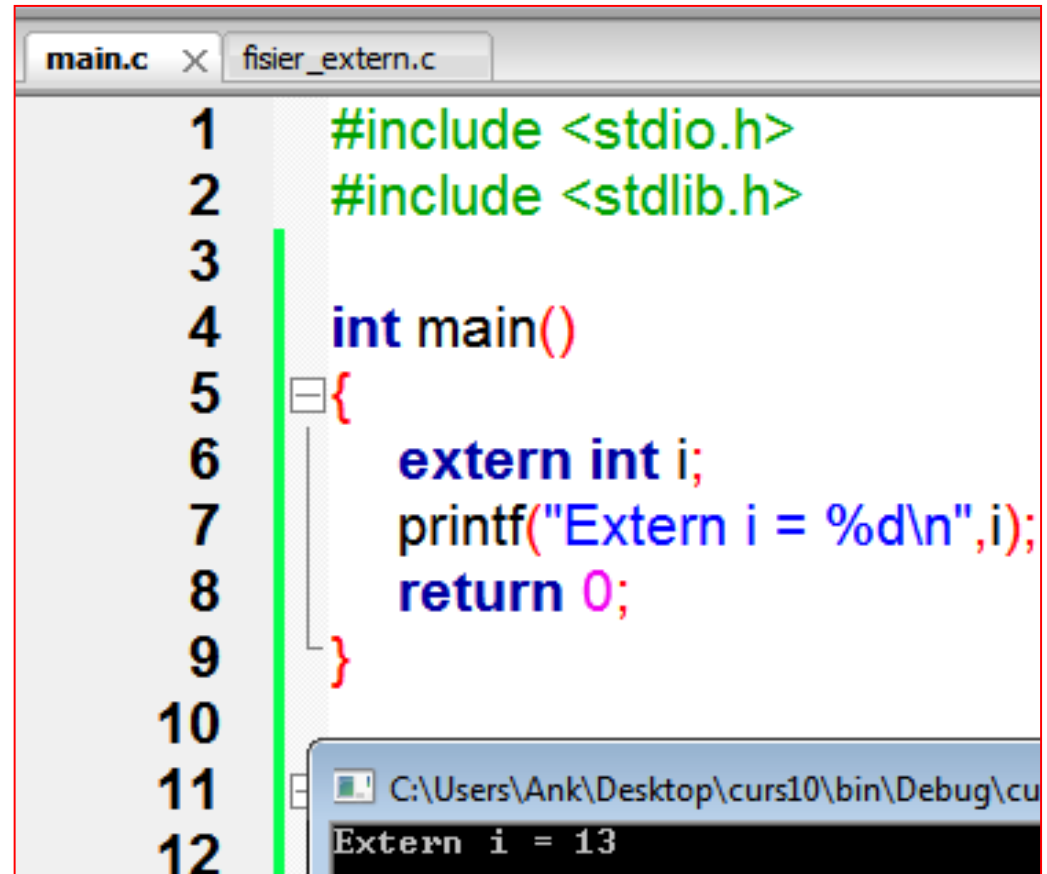
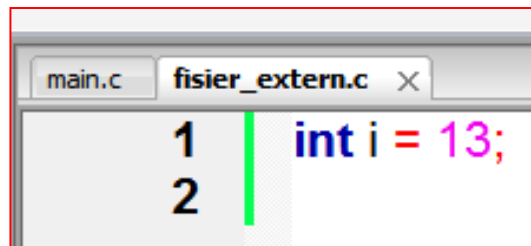
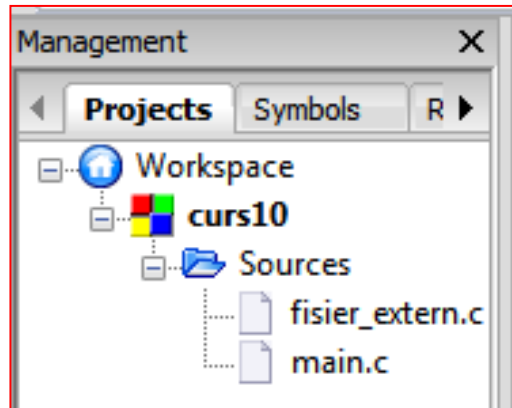
O variabila (functie) poate fi declarata in mai multe module (cu extern), dar trebuie definita intr-un singur modul!



2. Clase de alocare / memorare

Clasa de alocare: **static extern**

Expl 1. – acces la variabila definita in alt fisier





2. Clase de alocare / memorare

Clasa de alocare: **static extern**

Expl 2. – acces la variabila globala

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

int x = 10;

void f()
{
    int x = 33;
    {
        extern int x;
        printf("Variabila globala = %d\n\n", x);
    }
}

int main()
{
    f();
    return 0;
}
```

C:\Users\Ank\Desktop\curs10\bin\D
Variabila globala = 10



Agenda cursului

0. Alocarea dinamica a memoriei (recapitulare)
1. Funcții predefinite pentru manipularea blocurilor de memorie
 - memcpy, memmove, memchr, memcmp, etc
2. Clase de alocare / memorare
3. Subprograme recursive
4. Fisiere text si fisiere binare



3. Subprograme recursive

Recursivitate este proprietatea functiilor de a se autoapela.

- studiem mecanismul recursivității, ce se întâmplă când o funcție se auto-apelează (**nu studiem recursivitatea ca tehnică de programare**);
- corespondentul din matematică al recursivității este **recurența**



3. Subprograme recursive

Exemplu – factorialul unui numar natural

Definitii

$$n! = \begin{cases} 1, & \text{dacă } n=0 \\ n*(n-1)!, & \text{dacă } n \geq 1 \end{cases}$$

$$\begin{array}{l} 4! = 4*3! = 4 * 6 = 24 \\ 3! = 3*2! = 3 * 2 = 6 \\ 2! = 2*1! = 2 * 1 = 2 \\ 1! = 1*0! = 1 * 1 = 1 \\ 0! = 1 \end{array}$$

Adâncimea
recursivității

Definitie inutila (nu se opreste relatia de recurenta

$$n! = \frac{(n+1)!}{n+1}$$



3. Subprograme recursive

Sintaxa

```
tip functie_rekursiva (parametru formal)
{ ...
  conditie de oprire
  ramura de continuare
  functie_rekursiva (parametru formal modificat)
}
```

Toate instructiunile din subprogram se executa de cate ori este apelata functia.



3. Subprograme recursive

Exemplu – factorialul unui numar natural

$$n! = \begin{cases} 1, & \text{dacă } n=0 \\ n*(n-1)!, & \text{dacă } n \geq 1 \end{cases}$$

```
int factorial(int n)
{
    if (n==0) return 1; //conditia de oprire
    return n*factorial(n-1); //recursivitate
}
```

Ce se întâmplă în stivă pentru apelul $t = \text{factorial}(4)$?

În stivă, fiecare apel se așează deasupra apelului precedent.

Se salvează un context de apel.



3. Subprograme recursive

Exemplu – factorialul unui numar natural

Se salvează un context de apel:

1. **adresa de revenire**
2. **copii ale valorile parametrilor efectivii**
3. **valorile variabilelor locale**
4. **copii ale regiștrilor**
5. **valoarea returnată**

STIVĂ

A ₅	0	-	-	1
A ₄	1	-	-	1
A ₃	2	-	-	2
A ₂	3	-	-	6
A ₁	4	-	-	24

t = factorial(4);

↑
A₁

Adresa
de
revenire

Valoare
parametri
efectivii

Valoare
Variabile
locale

Regiștri

Valoare
returnată

Sursa: Alexe B – Programare Procedurala (Note de curs 2015)



3. Subprograme recursive

Orice functie recursiva trebuie sa contina **o conditie de oprire** respectiv, de continuare.

La fiecare reapel al functiei se executa aceeas secventa de instructiuni.

La fiecare reapel, in zona de stiva a memoriei:

- se ocupa un nivel nou
- se memoreaza valoarea parametrilor formali transmisi prin valoare
- adresa parametrilor formali transmisi prin referinta
- adresa de revenire
- variabilele cu valorile din momentul respectiv



3. Subprograme recursive

Obs:

- Toate instructiunile din subprogram se executa pentru fiecare reapel
- se executa instructiunile din functie pana la instructiunea de reapel
 - se executa din nou aceeasi secventa de instructiuni pana la conditia de oprire
 - procedeul se reia pana la intalnirea conditiei de oprire

Pentru fiecare apel s-a salvat in stiva un nivel, apoi pentru fiecare dintre aceste apeluri se executa instructiunile ramase in functie cu valoarea datelor din varful stivei **(atentie! vor fi in ordine inversa introducerii lor in stiva).**



3. Subprograme recursive

Expl. – suma primelor n numere naturale

```
int fun1(int n)
{
    if (n == 0) return 0;
    else return n + fun1(n-1);
}

// varianta iterativa
int fun2(int n)
{
    int y = 0;
    while (n!=0)
    {
        y = y + n;
        n--;
    }
    return y;
}
```

```
int main()
{
    int n;
    scanf("%d",&n);
    printf("Rezultat functie recursiva = %d\n", fun1(n));
    printf("Rezultat functie iterativa = %d\n", fun2(n));

    return 0;
}
```

```
"C:\Users\Ank\Desktop\Curs 9\bin\Debug\Curs 9.exe"
5
Rezultat functie recursiva = 15
Rezultat functie iterativa = 15
```



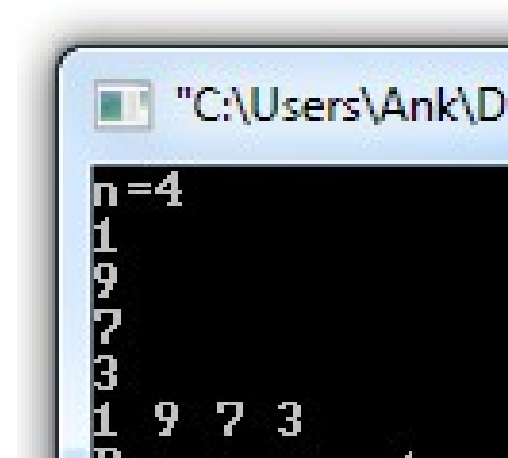
3. Subprograme recursive

Expl. – citirea si afisarea unui vector

```
void citire(int a[20],int n)
{
    scanf("%d",&a[n]);
    if(n>1) citire(a,n-1);
}

void afisare(int a[20],int n)
{
    if (n>=1)
    {
        printf("%d ",a[n]);
        afisare(a,n-1);
    }
}
```

```
int main()
{
    int a[20],n;
    printf("n="); scanf("%d",&n);
    citire(a,n);
    afisare(a,n);
}
```



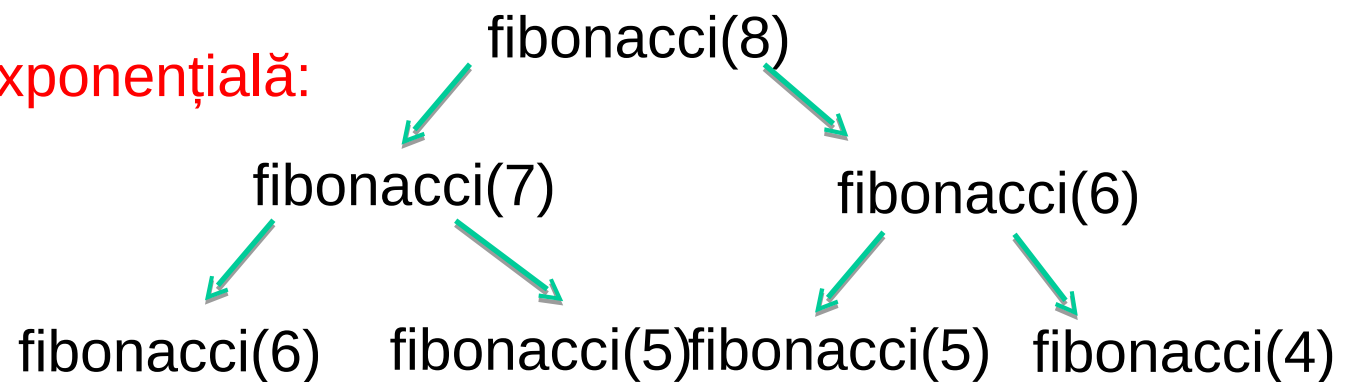


3. Subprograme recursive

Expl. – sirul lui Fibonacci: $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$, dacă $n \geq 2$

```
int fibonacci(int n)
{
    if (n <= 1)
        return n;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

Complexitate exponențială:





3. Subprograme recursive

Expl. – sirul lui Fibonacci: $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$, dacă $n \geq 2$

```
int fib_rec (int x)
{
    static int nr_apeluri = 0;
    nr_apeluri++;
    printf("Apel functie nr %d \n", nr_apeluri);

    if (x <= 2) return 1;
    return fib_rec(x-1) + fib_rec(x-2);
}

int main()
{
    int x = fib_rec(20);
    printf("x = %d",x);
    return 0;
}
```

```
Apel functie nr 13525
Apel functie nr 13526
Apel functie nr 13527
Apel functie nr 13528
Apel functie nr 13529
x = 6765
Process returned 0 (0x0)   execution time : 2.012 s
Press any key to continue.
```



3. Subprograme recursive

Expl. – sirul lui Fibonacci: $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$, dacă $n \geq 2$

```
int fib_iter(int x)
{
    int i,a,b,c;
    if (x <= 2) return 1;
    a = b = 1;
    for (i = 2; i < x; i++)
    {
        c = a + b;
        a = b;
        b = c;
    }
    return c;
}
```

Iterativ

```
int main()
{
    int x = fib_iter(20);
    printf("x = %d",x);
    return 0;
}
```

```
x = 6765
Process returned 0 (0x0)   execution time : 0.281 s
Press any key to continue
```




Agenda cursului

0. Alocarea dinamica a memoriei (recapitulare)
1. Funcții predefinite pentru manipularea blocurilor de memorie
 - memcpy, memmove, memchr, memcmp, etc
2. Clase de alocare / memorare
3. Subprograme recursive
4. Fisiere text si fisiere binare



4. Fisiere text si fisiere binare

fișier = șir de octeți (colecție de date) memorat pe suport extern (magnetic, optic) și identificat printr-un nume.

Expl: programe sursa (.c, .cpp), executabile, imagini (.jpeg, .jpg, .png, .bmp), documente (.pdf, .dvi, .eps), audio (.mp3), video: (.avi, .mp4).

Pentru fiecare tip de fisier binar, este necesar un program care sa interpreteze corect datele continute.

fișier = flux de date (stream) = transfer de informație binară (șir de octeți) de la o sursă spre o destinație:

- citire: flux de la tastatură (sursă) către memoria internă (destinație)
- afișare: flux de la memoria internă (sursă) către periferice (monitor, imprimantă)



4. Fisiere text si fisiere binare

Fluxuri automate asociate unui program

- **stdin (standard input)** – flux de intrare (citire).
 - asociat implicit cu tastatura.
- **stdout (standard output)** – flux de ieşire (afişare).
 - asociat implicit cu ecranul.
- **stderr (standard error)** – flux de ieşire (afişare) pentru erori.
 - asociat implicit cu ecranul.



4. Fisiere text si fisiere binare

Tipuri de fisiere

- **Text** – Un fișier text se accesează ca o succesiune de linii de text de lungime variabilă (incheiata cu ‘\n’) utilizând un set dedicat de funcții din biblioteca standard

Caracterele terminatorii de linii sunt: (LF – line feed – ‘\n’: Unix; CR carriage return – ‘\r’: Mac OS vechi; CR+LF – ‘\r\n’: Windows; caracter terminator de fisier EOF – CTRL-Z);

- **Binare** – Un fișier binar se accesează ca o succesiune de octeți, cărora funcțiile de citire și scriere din fișier nu le dau nici o interpretare.



4. Fisiere text si fisiere binare

Lucrul cu fisiere

Conceptul de bază “**Pointer la fisier**”

În fișierul `stdio.h` este definit un tip de structură, numit **FILE** care conține informații referitoare la un fișier: nume, adresă, adresă bufferului intern în care se procesează (citire/scriere) octeții din fișier, indicator de sfârșit de fișier, indicator de poziție în fișier, etc

Lucrul cu fisiere implica declararea unui pointer la fisier in vederea realizării legăturii dintre nivelul logic (variabila fișier) și nivelul fizic (numele extern al fișierului)

FILE * <identificator>;



4. Fisiere text si fisiere binare

Lucrul cu fisiere

Operatii:

1. Deschidere → SO aloca resurse interne pentru a se putea accesa continutul
2. Citirea / Scrierea conținutului
3. Inchidere → SO elibereaza resursele utilizate anterior.

Deschidere:

1. numele fișierului;
2. modul de accesare (citire, scriere, adaugare, actualizare - citire și scriere);

Distinctie intre fisierele binare si fisierele text → functii de biblioteca

3. tipul fișierului: text sau binar;



4. Fisiere text si fisiere binare

Lucrul cu fisiere

Functii uzuale

Nume	Scop
fopen()	Deschide un fișier
fclose()	Închide un fișier
putc()	Scrive un caracter într-un fișier
fputc()	La fel ca putc()
getc()	Citește un caracter dintr-un fișier
fgetc()	La fel ca getc()
fseek()	Caută un anumit octet într-un fișier
fprintf()	Este pentru un fișier ceea ce este printf() pentru consolă
fscanf()	Este pentru un fișier ceea ce este scanf() pentru consolă
feof()	Returnează adevărat dacă se ajunge la sfârșitul fișierului
ferror()	Returnează adevărat dacă a apărut o eroare
rewind()	Readuce indicatorul de poziție al fișierului la început
remove()	Șterge un fișier
fflush()	Golește un fișier

Sursa: Schildt H – C++ Manual complet (Teora, 1998)



4. Fisiere text si fisiere binare

Lucrul cu fisiere

Deschiderea unui fisier

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

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

```
FILE *fp;  
fp = fopen("test", "w");
```

```
FILE *p;  
if ((fp = fopen("test", "w"))==NULL) {  
    printf("Nu pot deschide fisierul.\n");  
    exit(1);  
}
```

Sursa: Schildt H – C++ Manual complet (Teora, 1998)



4. Fisiere text si fisiere binare

Lucrul cu fisiere

int *fclose(FILE *f)

Inchiderea unui fisier

Detectarea sfarsitului de fisier

int feof(FILE *f)

f = pointer la structura **FILE** corespunzătoare fișierului prelucrat.

- funcția **feof** returnează 0 dacă nu s-a ajuns la sfârșitul fișierului la ultima operație de citire sau o valoare nenulă in caz contrar

ATENȚIE:

```
while (!feof(f))  
    citeste x din f;  
    scrie x
```

se scrie un x (= -1) in plus



varianta corecta

```
while (1)  
    citeste x din f;  
    if (feof(f)) break;  
    scrie x
```



4. Fisiere text si fisiere binare

Lucrul cu fisiere

- stergere

int **remove**(const char * nume_fis);

- redenumire

int **rename**(const char *f_vechi, const char *f_nou);

Alte operatii la nivel de fisier

Ambele functii întorc o valoare nenulă în caz de eroare si zero în cazul normal.

- informatii despre un fisier - functiile **stat** si **fstat** cu prototipul declarat în fisierul stat.h din catalogul **sys**.

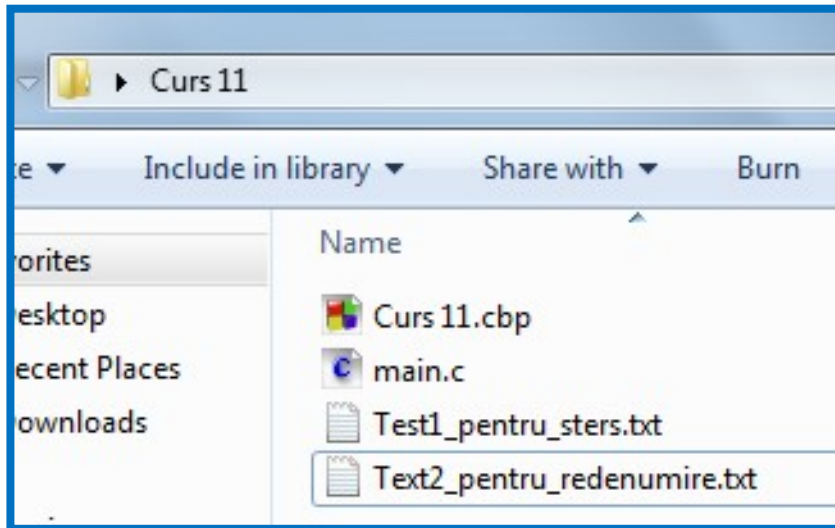
int **stat** (char *cale, struct stat * **buffer**);

buffer → adresa unei structuri de tip stat ale cărei câmpuri descriu starea entității în discutie (i.e câmpul st_size → dimensiunea unui fisier, în octeti.)

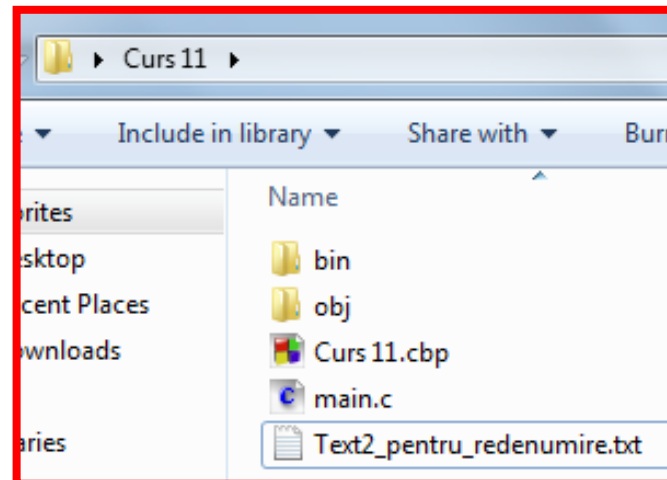
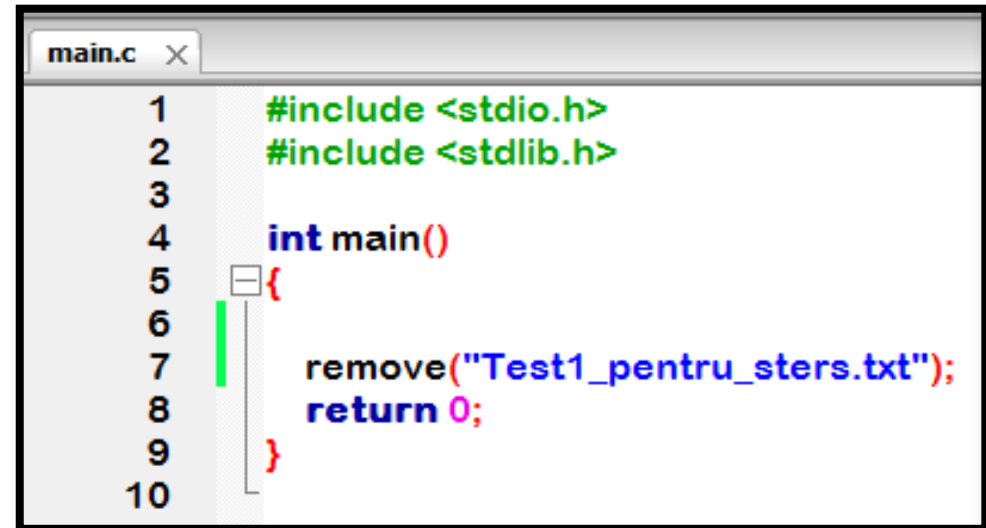


4. Fisiere text si fisiere binare

Lucrul cu fisiere



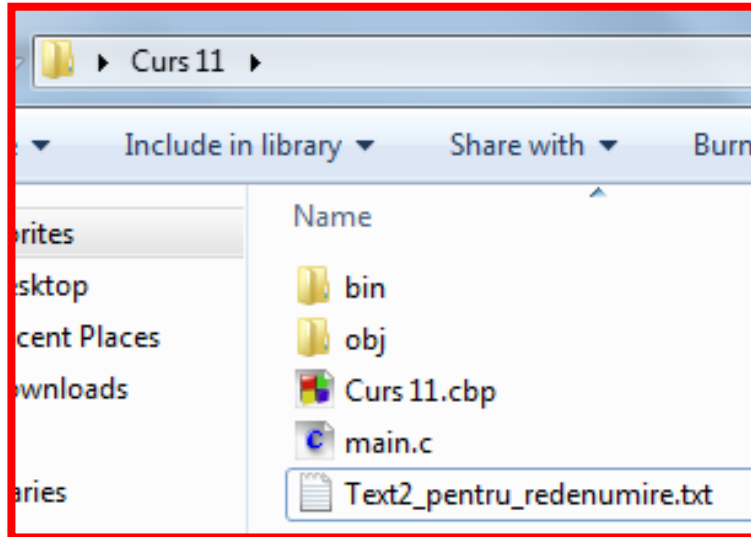
Testare operatii remove, rename



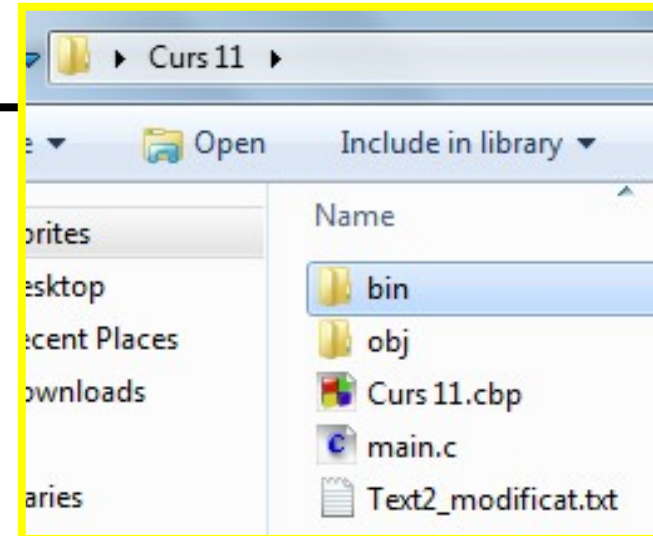
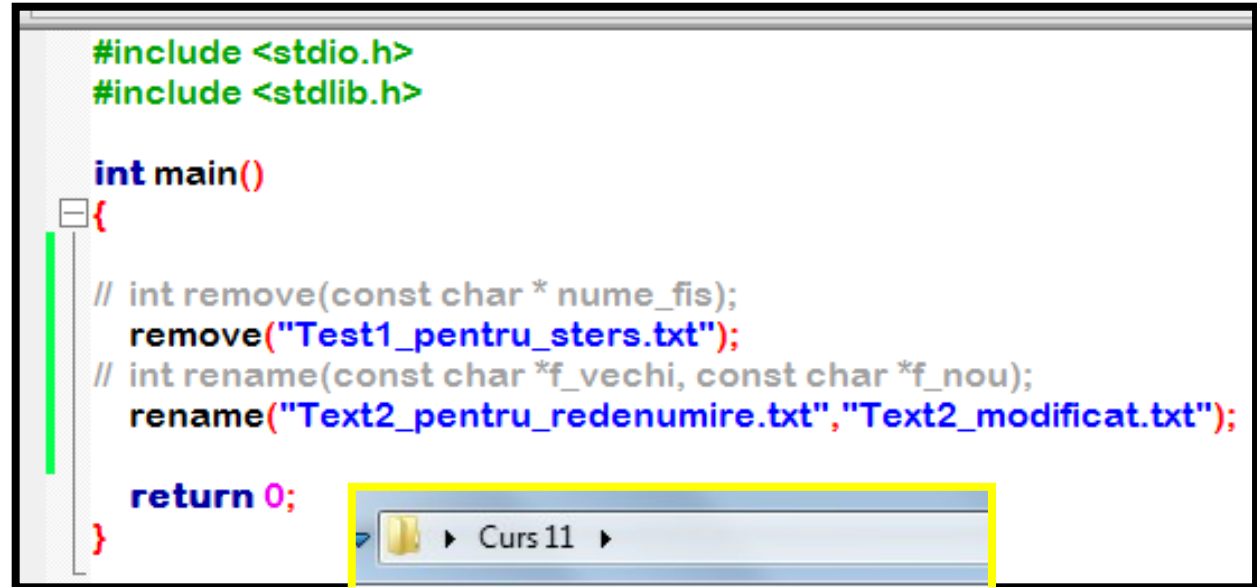


4. Fisiere text si fisiere binare

Lucrul cu fisiere



Testare operatii remove, rename





4. Fisiere text si fisiere binare

Lucrul cu fisiere

Testare operatii stat

```
struct stat
{
    dev_t      st_dev;      /* număr echipament */
    ino_t      st_ino;      /* număr inod */
    mode_t     st_mode;     /* tip fișier și permisiuni */
    nlink_t    st_nlink;    /* număr de legături hard */
    uid_t      st_uid;      /* UID proprietar */
    gid_t      st_gid;      /* GID proprietar */
    dev_t      st_rdev;     /* tip echipament */
    off_t      st_size;     /* mărime (în octeți) */
    blksize_t  st_blksize;  /* lungime bloc preferată pentru I/O */
    blkcnt_t   st_blocks;   /* numărul de blocuri de 512 octeți alocate */
    time_t     st_atime;    /* timpul ultimului acces */
    time_t     st_mtime;    /* timpul ultimei modificări */
    time_t     st_ctime;    /* timpul ultimei schimbări a stării – scrierea
                             sau setarea informațiilor din i-nod-uri */
};
```



4. Fisiere text si fisiere binare

Lucrul cu fisiere

Testare operatii stat

```
struct stat
{
    dev_t      st_dev;      /* număr echipament */
    ino_t      st_ino;      /* număr inod */
    mode_t     st_mode;     /* tip fișier și permisiuni */
    nlink_t    st_nlink;    /* număr de legături hard */
    uid_t      st_uid;      /* UID proprietar */
    gid_t      st_gid;      /* GID proprietar */
    dev_t      st_rdev;     /* tip echipament */
    off_t      st_size;     /* mărime (în octeți) */
    blksize_t  st_blksize;  /* lungime bloc preferată pentru I/O */
    blkcnt_t   st_blocks;   /* numărul de blocuri de 512 octeți alocate */
    time_t     st_atime;    /* timpul ultimului acces */
    time_t     st_mtime;    /* timpul ultimei modificări */
    time_t     st_ctime;    /* timpul ultimei schimbări a stării – scrierea
                             sau setarea informațiilor din i-nod-uri */
};
```

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>

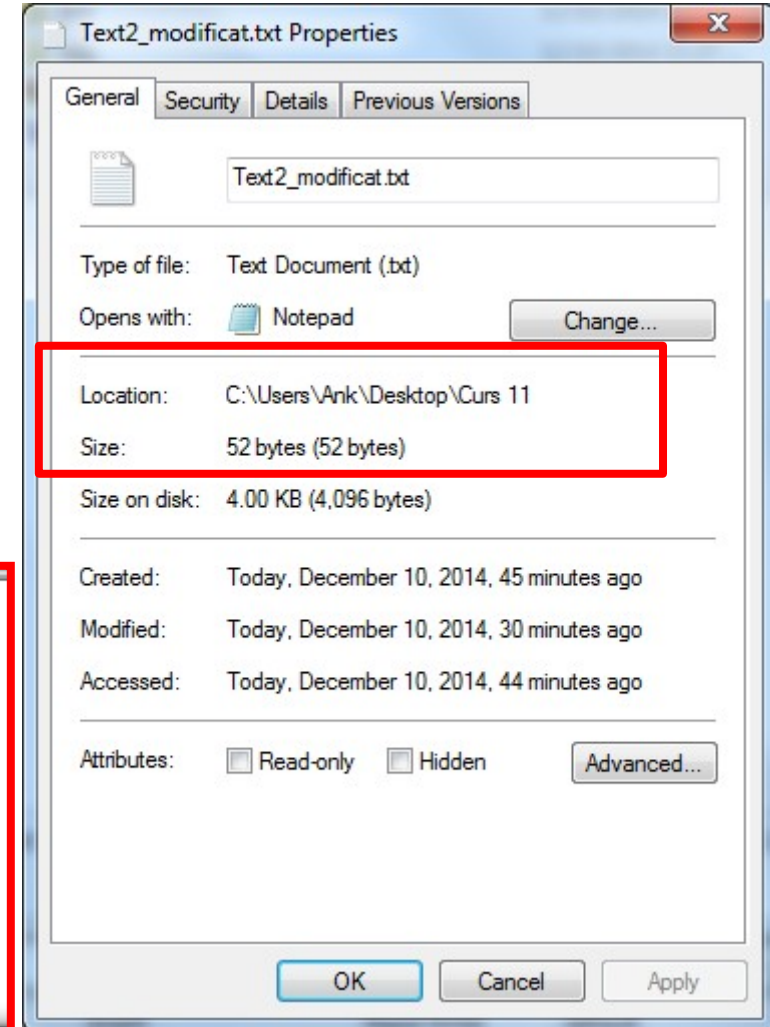
int main()
{
    struct stat buffer;

    stat("Text2_modificat.txt",&buffer);
    printf("\n Marimea fisierului pe disc este %d \n\n",buffer.st_size);

    return 0;
}
```

"C:\Users\Ank\Desktop\Curs 11\bin\Debug\Curs 11.exe"

Marimea fisierului pe disc este 52





4. Fisiere text si fisiere binare

Fisiere text – functii specifice de manipulare

- ❑ accesul la fişierele text se poate face la nivel de **şir de caractere** (linie) sau la nivel de **caracter** (octet).
- ❑ 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.
- ❑ funcţiile de citire sau de scriere cu format din/în fişiere text realizează conversia automată din:
 - ❑ format extern (şir de caractere) în format intern (binar) - la citire
 - ❑ format intern (binar) în format extern (şir de caractere), la scriere pentru numere întregi sau reale

Sursa: Alexe B – Programare procedurala (Note de curs 2016)



4. Fisiere text si fisiere binare

Fisiere text – functii specifice de manipulare

- `int fgetc(FILE *f)` întoarce codul ASCII al caracterului citit din fișierul `f` sau `-1` (EOF) în caz de eroare;
- `int fputc(int c, FILE *f)` scrie caracterul cu codul ASCII `c` în fișierul `f` sau `-1` (EOF) în caz de eroare;
- `char* fgets(char *sir, int m, FILE *f)` citește maxim `m-1` caractere sau până la `'\n'` și pune șirul de caractere în `sir` (adaugă la sfârșit `'\0'`).
 - returnează adresa șirului citit sau `NULL` în caz de eroare
- `int fputs(char *sir, FILE *f)` scrie șirul `sir` în fișierul `f`, fără a pune `'\n'` la sfârșit.
 - întoarce numărul de caractere scrise, sau `EOF` în caz de eroare.



4. Fisiere text si fisiere binare

Fisiere text – functii specifice de manipulare

Utilizarea functiilor fgets(), fputs()

	main.c	Date.in	Date.out
1		Propozitie pe linia 1	
2		Propozitie pe linia 2	
3		Propozitie pe linia 3	
4			

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

int main()
{
    FILE *f,*g;
    char s[101];
    f = fopen("Date.in","r");
    g = fopen("Date.out","w");

    fgets(s,100,f);
    fputs(s,g);

    return 0;
}
```

	main.c	Date.in	Date.out
1			Propozitie pe linia 1
2			



4. Fisiere text si fisiere binare

Fisiere text – functii specifice de manipulare

Utilizarea functiilor fgets(), fputs(), rewind()

```
main.c Date.in X Date.out
1 Propozitie pe linia 1
2 Propozitie pe linia 2
3 Propozitie pe linia 3
4
```

```
FILE *f,*g;
char s[101];
f = fopen("Date.in","r");
g = fopen("Date.out","w");

fgets(s,100,f);
fputs(s,g);
fgets(s,100,f);
fputs(s,g);
fgets(s,100,f);
fputs(s,g);

rewind(f);

fgets(s,100,f);
fputs(s,g);
```

```
main.c Date.in X Date.out
1 Propozitie pe linia 1
2 Propozitie pe linia 2
3 Propozitie pe linia 3
4 Propozitie pe linia 1
5
```



4. Fisiere text si fisiere binare

Fisiere text – functii specifice de manipulare

Utilizarea functiilor fopen() fgetc(), fputc(), feof(), fclose()

Exemplul 1 (Copierea unui fișier de tip text f1.txt în fișierul text f2.txt).

```
#include <stdio.h>
int main(void) {
FILE *sursa, *dest;
if ((sursa=fopen("f1.txt", "rt")) == NULL) {
    fprintf(stderr, "Nu se poate deschide f1.txt !");
    return 1;
}
if ((dest=fopen("f2.txt", "wt")) == NULL) {
    fprintf(stderr, "Nu se poate deschide f2.txt!");
    return 2;
}
while (!feof(sursa)) fputc(fgetc(sursa), dest);
fclose(sursa);
fclose(dest);
return 0;
}
```

Sursa: Albeanu G. – Programare procedurala. Suport de curs 2013.



4. Fisiere text si fisiere binare

Fisiere text – functii specifice de manipulare

Functii de citire / scriere cu format

- ❑ `int fscanf(FILE *f, char *format)`
 - ❑ citește din fisierul `f` folosind un format (analog cu `scanf`)
- ❑ `int fprintf(FILE *f, char *format)`
 - ❑ scrie în fișierul `f` folosind un format (analog cu `printf`)



4. Fisiere text si fisiere binare

Fisiere text – functii specifice de manipulare

Utilizarea functiilor fprintf(), fscanff()

```
int main()
{
    FILE *f,*g;
    char s[101];
    f = fopen("Date.in","r");
    g = fopen("Date.out","w");

    printf("Se citeste de la tastatura: \n");
    fscanf(stdin, "%s", s);
    fprintf(stdout, "Se afiseaza pe ecran: %s \n", s);
    printf("Se citeste din fisier: \n");
    fgets(s, 100, f);
    fprintf(stdout, "Se afiseaza pe ecran: %s \n", s);
}
```

```
"C:\Users\Ank\Desktop\Curs 10\bin\Debug\Curs 10.exe"
Se citeste de la tastatura:
Sir_test
Se afiseaza pe ecran: Sir_test
Se citeste din fisier:
Se afiseaza pe ecran: Propozitie pe linia 1
```

main.c	Date.in	Date.out
1		Propozitie pe linia 1
2		Propozitie pe linia 2
3		Propozitie pe linia 3
4		



4. Fisiere text si fisiere binare

Fisiere binare – functii specifice de manipulare

Notiunea fundamentala → zonă compactă de octeti **(înregistrare)** care se citește sau se scrie.

-un fișier binar este format în general din articole de lungime fixă, fără separatori între articole. Un articol poate conține:

- un singur octet
- un număr binar (pe 2, 4 sau 8 octeți)
- structură cu date de diferite tipuri

Citirea / scrierea se face de la / la pozitia curentă din fisier.

→ Dupa executarea operatiei, pozitia indicatorului (de octet în cazul limbajului C) este actualizată automat, pentru a indica următoarea înregistrare.



4. Fisiere text si fisiere binare

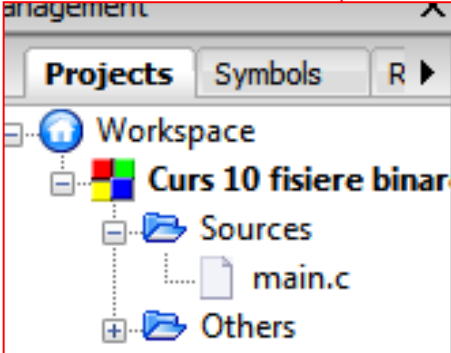
Fisiere binare – functii specifice de manipulare

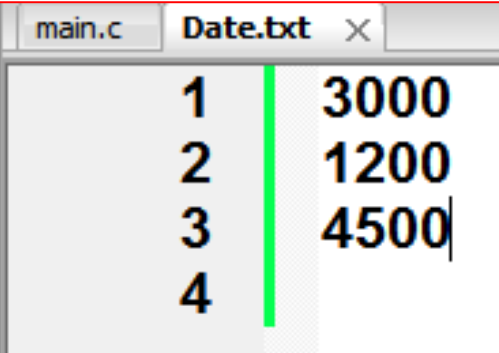
Utilizarea unui fisier text ca fisier binar

```
int main()
{
    FILE *f = fopen("Date.txt","rb");
    int c;

    while((c = fgetc(f))!= EOF)
        printf("%d ",c);

    printf("\n\n");
    return 0;
}
```

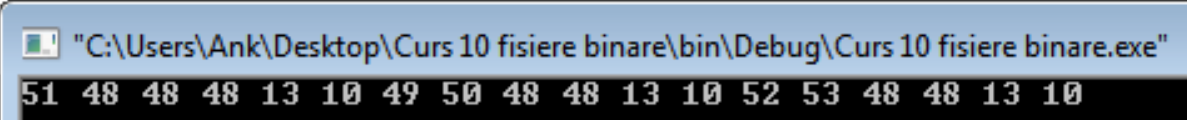




Ce observam?

- 51 – ASCII(3);
- 48 – ASCII(0);
- 13 – ASCII(CR – '\r');
- 10 – ASCII(LF – '\n');

Testat sub Windows





4. Fisiere text si fisiere binare

Fisiere binare – functii specifice de manipulare

- ❑ `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 fișierul referit de `f` la adresa `tablou`.



4. Fisiere text si fisiere binare

Expl: - crearea unui fisier binar cu n numere intregi

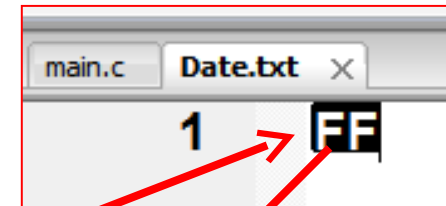
```
FILE *f,*g;
int n,x;
f = fopen("Date.txt","wb");

scanf("%d",&n);
while(n)
{
    scanf("%d",&x);
    fwrite(&x,sizeof(int),1,f);
    n--;
}
fclose(f);

g = fopen("Date.txt","rb");

while(fread(&x,sizeof(int),1,g)==1)
{
    printf("x = %d\n",x);
}
fclose(g);
```

```
3
12
23
34
x = 12
x = 23
x = 34
```



11	B	013	VT	(vertical tab)
12	C	014	FF	(NP form feed, new page)
13	D	015	CR	(carriage return)



4. Fisiere text si fisiere binare

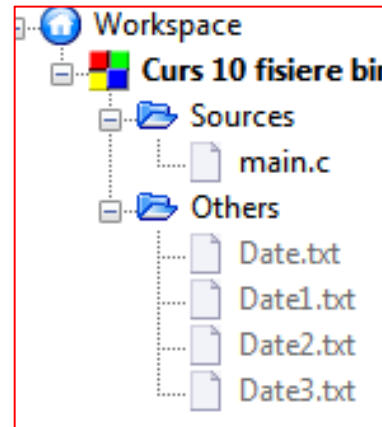
Expl: - crearea unui fisier binar cu numere intregi (modalitati echivalente)

```
int main()
{
    FILE *f,*g,*h;
    int i,v[5]={12,34,56,78,90};

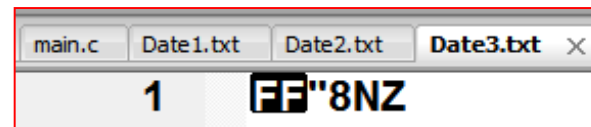
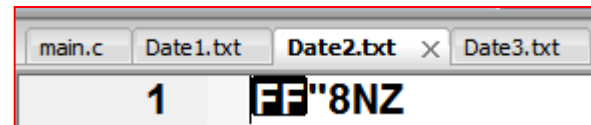
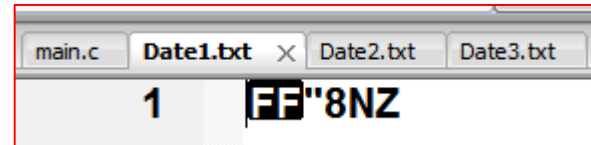
    f = fopen("Date1.txt","wb");
    for(i = 0; i<5; i++)
        fwrite(&v[i],sizeof(int),1,f);
    fclose(f);

    g = fopen("Date2.txt","wb");
    fwrite(v,sizeof(int),5,g);
    fclose(g);

    h = fopen("Date3.txt","wb");
    fwrite(v,5*sizeof(int),1,h);
    fclose(h);
}
```



12 – ASCII(FF);
34 – ASCII(“);
56 – ASCII(‘8’);
78 – ASCII(‘N’);
90 – ASCII(‘Z’)





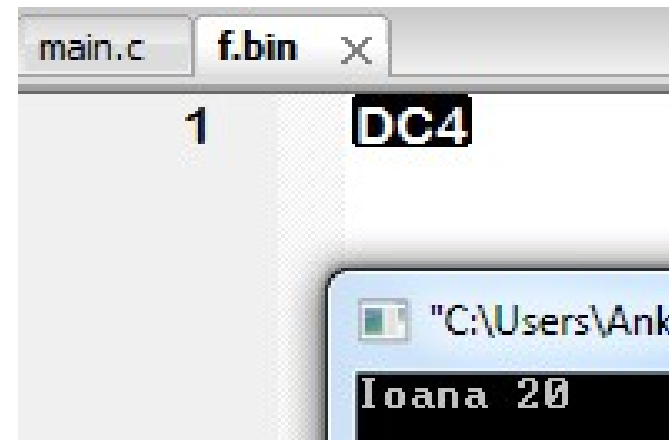
4. Fisiere text si fisiere binare

Expl: - crearea unui fisier binar cu 1 structura

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

int main()
{
    FILE *f;
    student st;

    f = fopen ("f.bin", "wb");
    if (f == NULL)
        printf ("Fisierul nu se poate crea!");
    else
    {
        scanf("%s", st.nume);
        scanf("%d", &st.varsta);
        /* scrierea structurii in fisier */
        fwrite (&st, sizeof(st), 1, f);
        fclose (f);
    }
}
```





4. Fisiere text si fisiere binare

Expl: - citirea dintr-un fisier binar anterior creat

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

int main()
{
    FILE *f;
    student st;
    int n;

    f = fopen ("f.bin", "rb");
    n = fread (&st, sizeof (st), 1, f);
    if (n == 1)
        printf ("%s are %d ani \n", st.nume, st.varsta);
    else
        printf ("Nu s-a putut citi inregistrarea din fis.");
    fclose (f);
    return 0;
}
```

"C:\Users\Ank\Desktop\Curs 1
Ioana are 20 ani



4. Fisiere text si fisiere binare

Expl: - crearea unui fisier binar cu n articole si afisarea continutului

```
int main()
{
    FILE *f, *g;
    student st[20];
    int n,i,nr;

    f = fopen ("f.bin", "wb");
    scanf("%d",&n);

    for ( i = 0; i < n; i++)
    {
        printf ("Numele persoanei %d: ", i + 1); scanf("%s",st[i].nume);
        printf ("Varsta persoanei %d: ", i + 1); scanf ("%d", &st[i].varsta);
    }
    fwrite (st, sizeof (student), n, f);
    fclose (f);

    g = fopen ("f.bin", "rb");
    nr = fread (st, sizeof (student), n, g);
    for ( i = 0; i < n; i++)
        printf ("Nume: %s, varsta %d: \n",st[i].nume,st[i].varsta);
    fclose (g);
    return 0;
}
```

```
3
Numele persoanei 1: Ioana
Varsta persoanei 1: 20
Numele persoanei 2: Alexandra
Varsta persoanei 2: 18
Numele persoanei 3: Andrei
Varsta persoanei 3: 24
Nume: Ioana, varsta 20:
Nume: Alexandra, varsta 18:
Nume: Andrei, varsta 24:
```



4. Fisiere text si fisiere binare

Expl: - manipularea cheii de sortare si a pozitiei unei structuri intr-un fisier

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

int main()
{
    student x,st[10];
    int n,i,j,index[10],aux;
    FILE *f,*g;

    f = fopen("Date.txt","wb");
    scanf("%d",&n);
    for(i = 0; i<n; i++)
    {
        scanf("%s%d",x.nume,&x.varsta);
        fwrite(&x,sizeof(x),1,f);
    }
    fclose(f);
}
```

Creare fisier

```
3
Ana 40
Ioana 20
Alex 30
Ioana 20
Alex 30
Ana 40
```



4. Fisiere text si fisiere binare

Expl: - manipularea cheii de sortare si a pozitiei unei structuri intr-un fisier

```
g = fopen("Date.txt","rb");
for (i = 0; i < n; i++)
{
    fread(&st[i],sizeof(student),1,g);
    index[i] = i;
}
fclose(g);

for (i = 0; i<n-1;i++)
    for (j = i+1; j <n; j++)
        if (st[index[i]].varsta > st[index[j]].varsta)
        {
            aux = index[i]; index[i] = index[j]; index[j] = aux;
        }

for (i = 0; i<n; i++)
    printf("%s\t%d\n",st[index[i]].nume, st[index[i]].varsta);
```

Ordonare articole

```
3
Ana 40
Ioana 20
Alex 30
Ioana 20
Alex 30
Ana 40
```




4. Fisiere text si fisiere binare

Fisiere binare – functii specifice de manipulare

Controlul indicatorului de pozitie

Citirea indicatorului de pozitie: **int fgetpos (FILE *f, long int *poz);**

Returneaza 0 (succes) si inscrie valoarea indicatorului in variabila poz.

Intoarcerea pozitiei curente in fisier: **long int ftell (FILE *f);**



4. Fisiere text si fisiere binare

Fisiere binare – functii specifice de manipulare

Controlul indicatorului de pozitie

Modificarea valorii indicatorului: **fsetpos, fseek si rewind.**

int fsetpos (FILE *p, const long int *poz);

Returneaza 0 (succes) si atribuie indicatorului valoarea poz.

Deplasarea cu nr_octeti relativ la o pozitie de referinta:

int fseek(FILE *f, long nr_octeti, int origine);

Returneaza 0 (succes) si nenula in caz de esec.

Valori utilizate: 0 = SEEK_SET (inceput de fisier)
1 = valoare curenta
2 = SEEK_END (sfarsit de fisier)



4. Fisiere text si fisiere binare

Expl: - aflarea dimensiunii unui fisier folosind functiile specifice indicatorului de pozitie

```
main.c x Text2_modificat.txt
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/stat.h>
4
5  int main()
6  {
7      FILE * f;
8      long int cp, lungime;
9
10     f = fopen("Text2_modificat.txt", "a");
11     fseek(f, 0L, SEEK_END);
12
13     lungime = ftell(f);
14     fprintf(f, "\n Marimea fisierului pe disc este %ld \n\n", lungime);
15
16     return 0;
17 }
18
```

Continut initial

Management X

Projects Symbol

Workspace

Curs 11

Sources

main.c Text2_modificat.txt X

```
1
2  Studiu de caz pentru cursul 11.
3
4  Despre fisiere.
```



4. Fisiere text si fisiere binare

Expl: - aflarea dimensiunii unui fisier folosind functiile specifice indicatorului de pozitie

```
main.c x Text2_modificat.txt
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/stat.h>
4
5  int main()
6  {
7      FILE * f;
8      long int cp, lungime;
9
10     f = fopen("Text2_modificat.txt","a");
11     fseek(f,0L,SEEK_END);
12
13     lungime = ftell(f);
14     fprintf(f,"\n Marimea fisierului pe disc este %ld \n\n", lungime);
15
16     return 0;
17 }
18
```

Continut actualizat

```
main.c Text2_modificat.txt x
1
2  Studiu de caz pentru cursul 11.
3
4  Despre fisiere.
5
6  Marimea fisierului pe disc este 56
7
```

Location: C:\Users\Ank\Desktop\Curs 11
Size: 56 bytes (56 bytes)
Size on disk: 4.00 KB (4,096 bytes)



Concluzii

1. S-a recapitulat notiunea de alocare dinamica;
2. S-au introdus notiunile de:
 - Clase de memorare
 - Recursivitate
3. S-au predat:
 - Funcții predefinite pentru manipularea blocurilor de memorie
 - Fisiere text si fisiere binare



Perspective

Cursul 11:

1. Structuri autoreferite

2. Functii cu numar variabil de argumente