Fundamentele programării

Curs 6

Pointeri

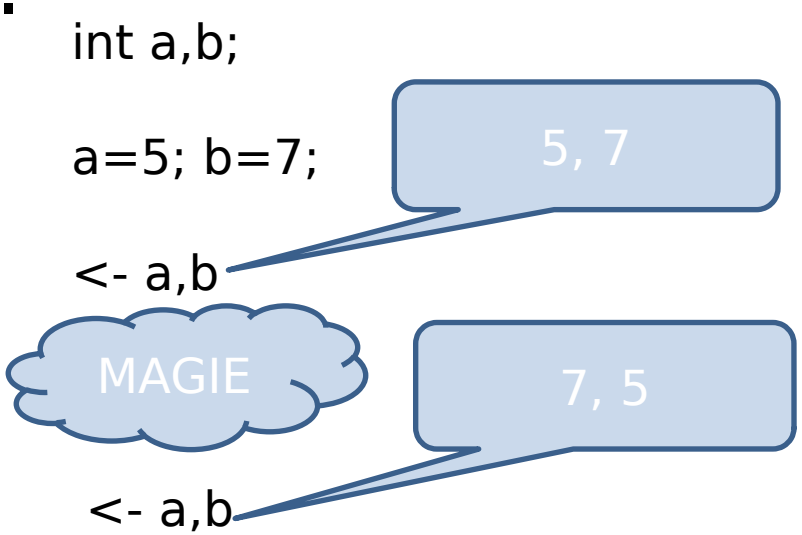
Elena BAUTU & Dorin-Mircea POPOVICI
{ebautu,dmpopovici}@univ-ovidius.ro

Cuprins

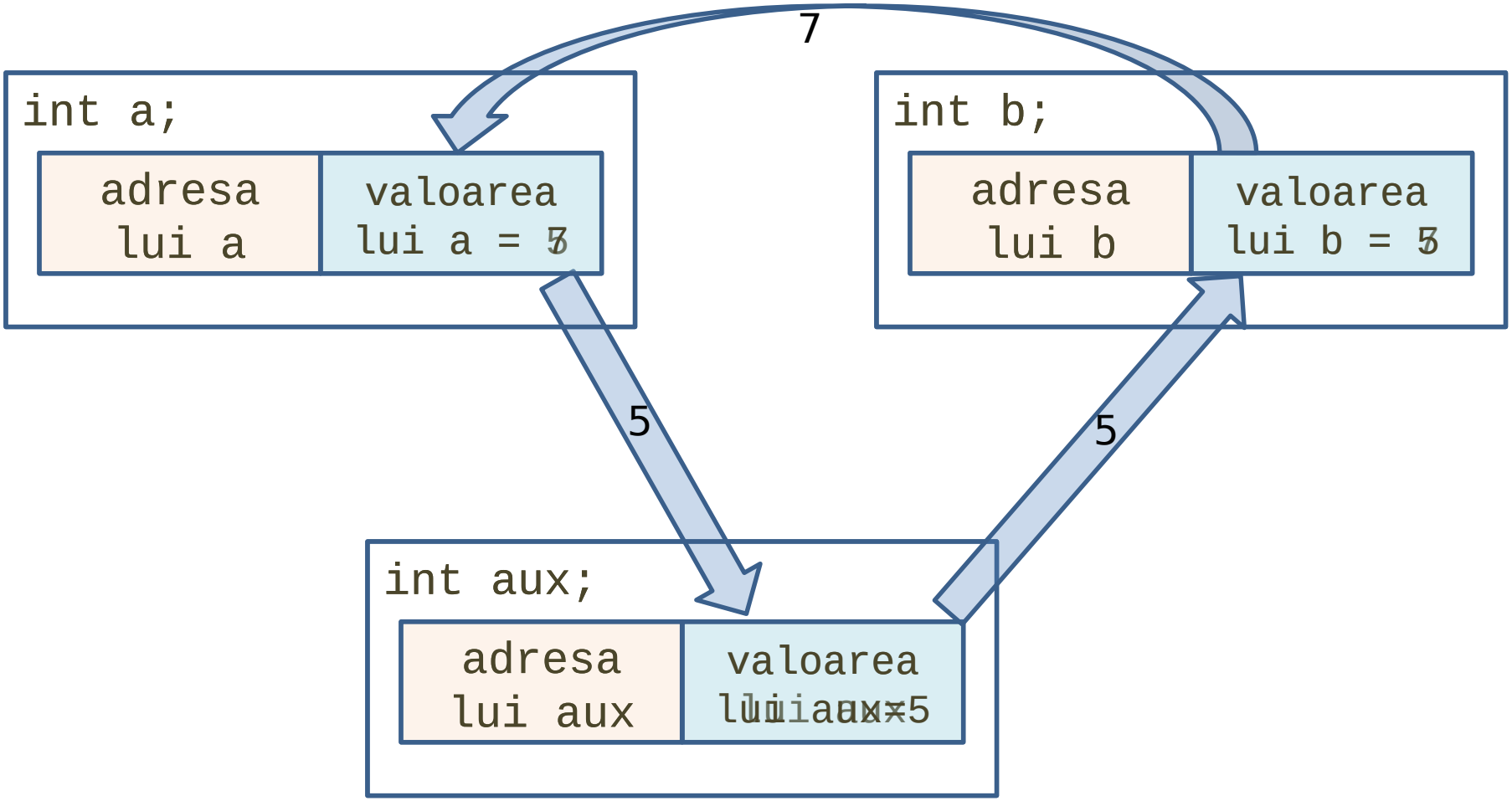
- Ce este un pointer?
- Transferul parametrilor
- Aritmetica pointerilor
- Accesul direct la masive de date
- Gestiunea dinamica a memoriei

Exercitiul 4.1

Sa se scrie o functie care interschimba valorile a doua variabile locale functiei **`main()`**.



Exercitiul 4.1 – Ideea



Exercitiul 4.1 – Pasul 1

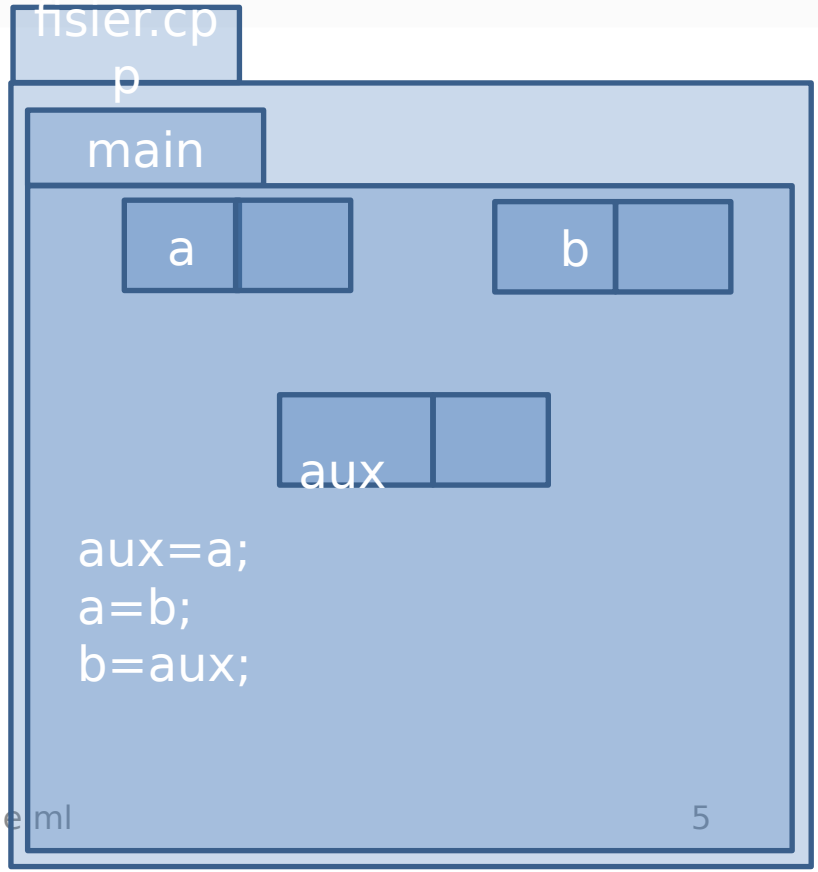
- Scriu un program care interschimba valorile celor doua variabile locale functiei main():

```
int main() {
  int a=5, b=7;
  int aux;

  printf("%d,%d",a,b);
  aux=a;
  a=b;
  b=aux;
  printf("%d,%d",a,b);
}
```

5, 7

MAGIE

7, 5



Exercitiul 4.1 – Pasul 2

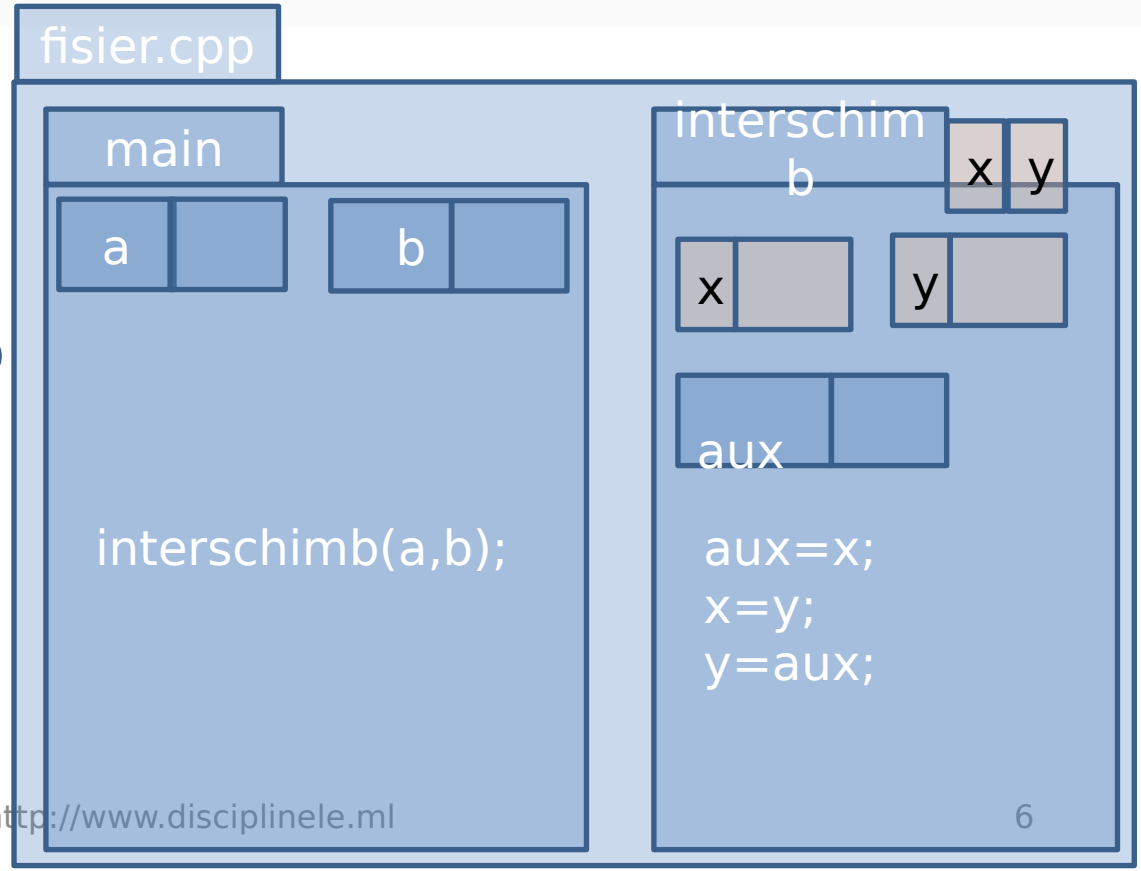
- Transform programul in functie, si ...

```
void interschimb(int,int);
int main() {
  int a=5, b=7;

  printf("%d, %d",a,b);
  interschimb(a,b);
  printf("%d, %d",a,b);
}
void interschimb(int x,int y){
  int aux;
  aux=x;
  x=y;
  y=aux;
}
```

5, 7

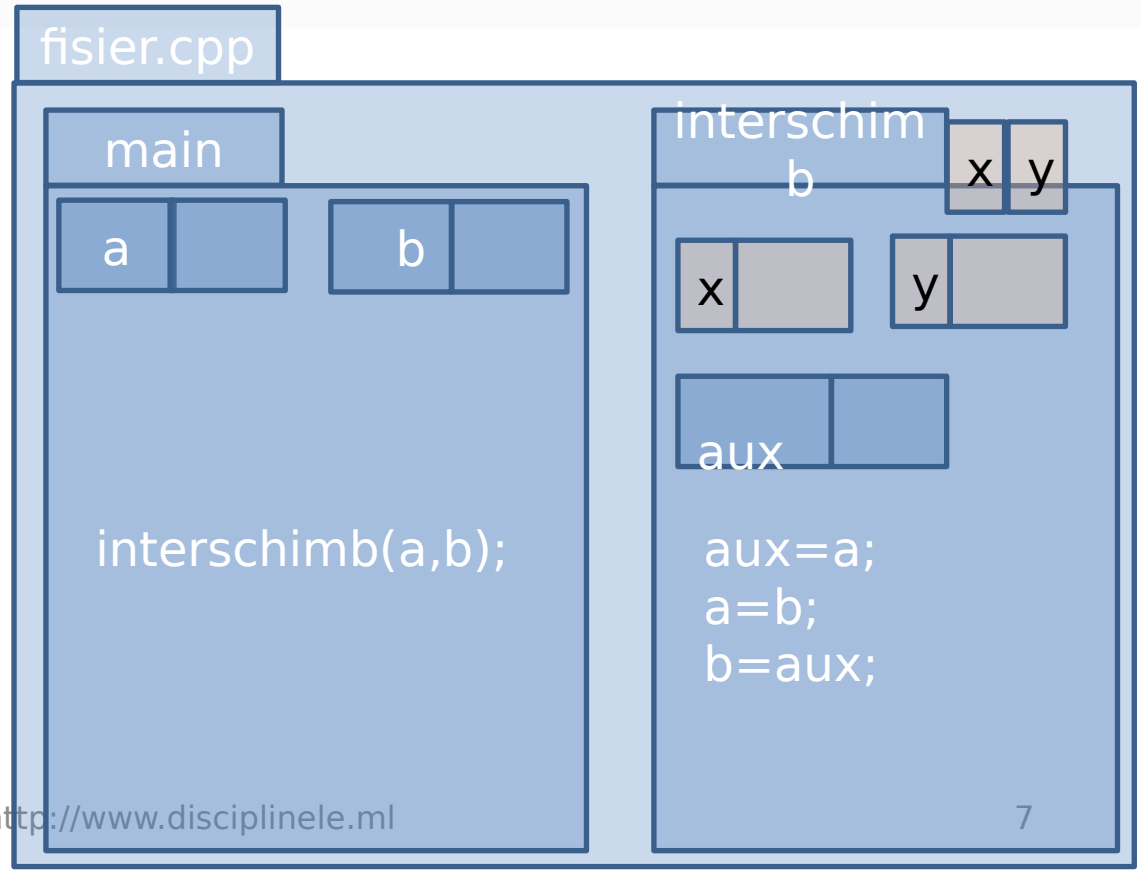
MAGIE

5, 7



Exercitiul 4.1 – Ce nu a mers????

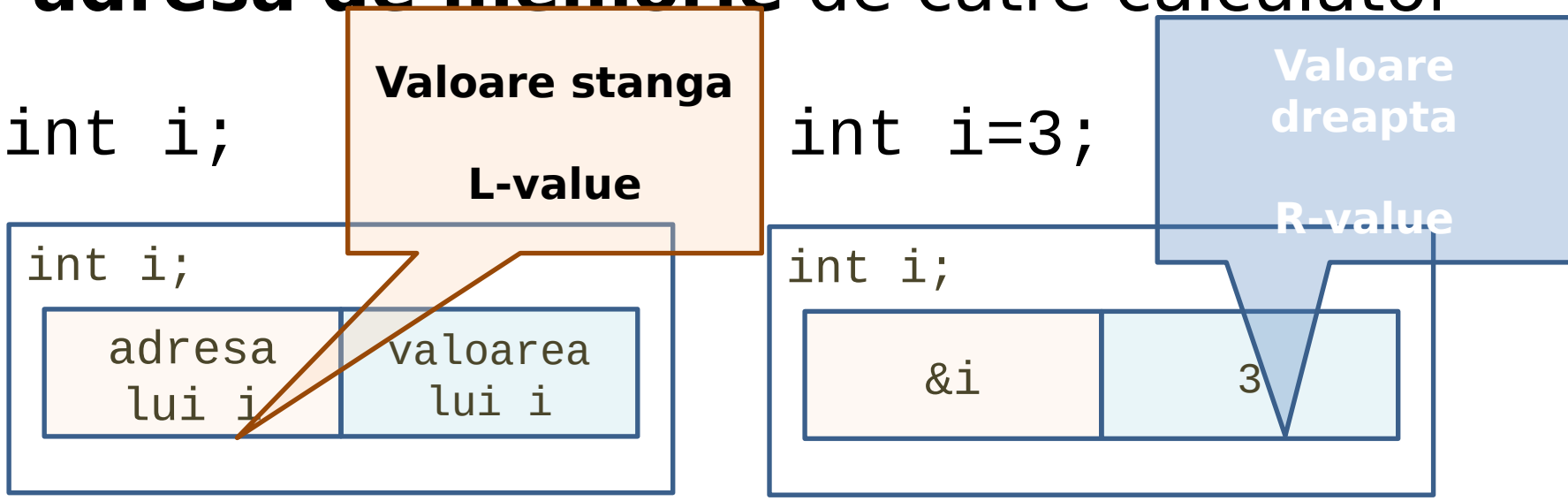
```
void interschimb(int,int)
int main() {
  int a=5, b=7;

  printf(“%d,%d”,a,b);
  interschimb(a,b);
  printf(“%d,%d”,a,b);
}
void interschimb(int x,int y){
  int aux;
  aux=x;
  x=y;
  y=aux;
}
```



Ce este un pointer?

Variabila - recall

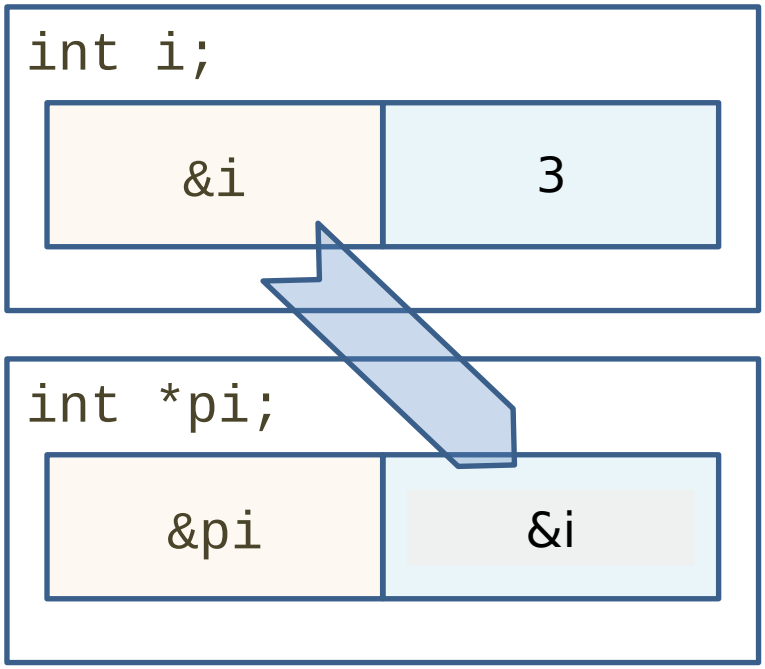
- Variabila = o **zona de memorie** capabila sa retina o informatie de un anumit **tip**, accesibila de catre programator printr-un **nume** si printr-o **adresa de memorie** de catre calculator



Pointerul

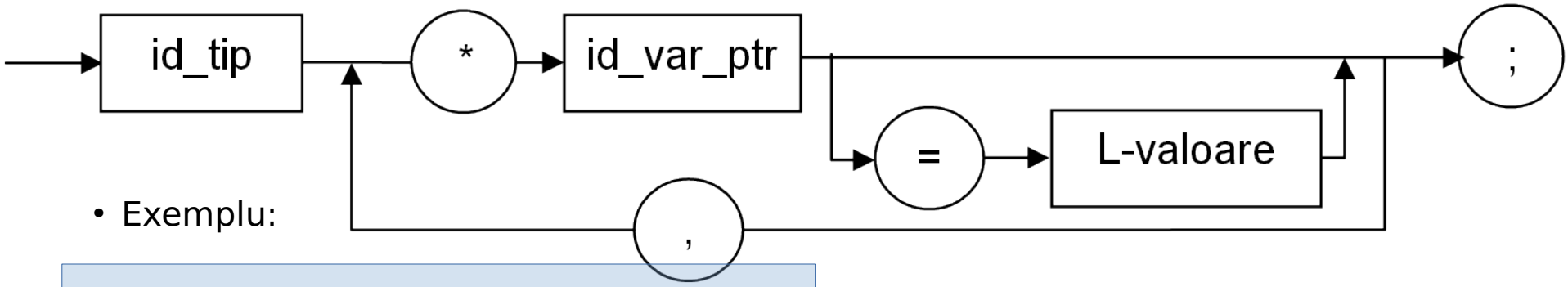
Un pointer este o variabila a carei valoare dreapta este o valoare stanga (i.e. pastreaza **ADRESA** unei date si **NU VALOAREA** datei).

```
int i;
i=3;

int *pi;
pi=&i;
```



Pointerul (cont)

- Declaratia unei variabile pointer:



- Exemplu:

```
int i;
int *pint=&i;
printf("Nr: %d, adresa lui: %p", i, pint);
```

```
float nr, *pnr;
pnr = &nr;
printf("Nr: %f, adresa lui: %p", nr, pnr);
```

- Operatorul ***** se numeste **operator de indirectare** (continut)
- Operatorul **&** se numeste **operator de adresare** (adresa)
- Sunt operatori unari, deci au **prioritate cea mai mare**.
- Pentru afisarea unui pointer in printf, se foloseste specificatorul de format **%p**

Exercitiul 4.2

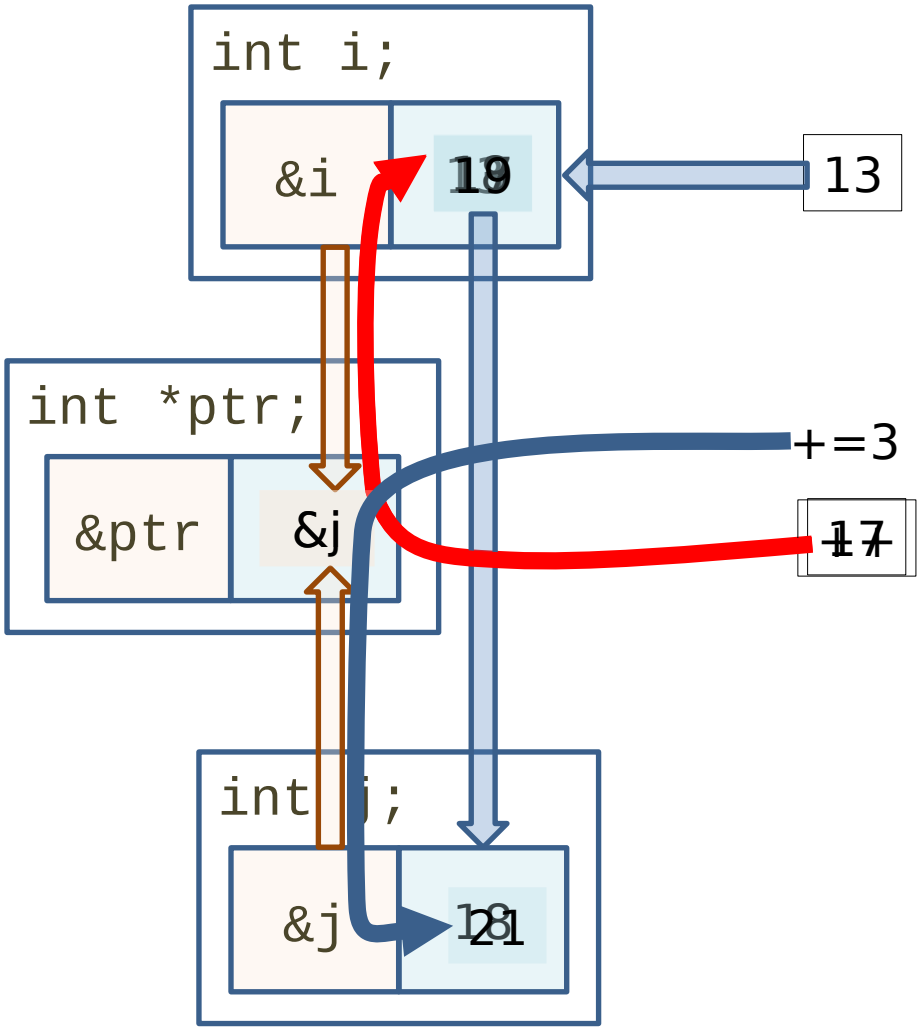
- Decideti corectitudinea instructiunilor de mai jos si spuneti ce valori sunt vehiculate:

- int x = 5;
  int *pi= &x;
  int y = *pi+5;
- *pi = 7;

- float z = 1;
  float *pf = &z;
  *pf = 2.5;
  printf(“%f\t%f\t%p”, z, *pf, pf); // ce afiseaza?

Pointerul (cont)

a. int i,j;
b. int *ptr;
c. ptr=&i;
d. i=13;
e. *ptr=17;
f. j=++i;
g. *ptr++;
h. ptr=&j;
i. *ptr+=3;



Exercitiul 4.3

- Decideti corectitudinea instructiunilor de mai jos si spuneti ce valori sunt vehiculate:

```
int i,*pi;
pi= &i;
i=13;
*pi=29;
pi=89;
i++:
(*pi)--;
```

- Indicatie: * => continut, & =>adresa, deci, daca pi=&i, atunci
- *pi <=> *&i (continutul de la adresa lui i)

Exercitiul 4.4

- Ce se tipareste pe ecran la executia secventei de mai jos:

```
double n = 5.6, *pn;
pn = &n;
*pn += 3;
printf("%lf    %ld    %ld\n", n, sizeof(n), sizeof(pn));
```

a) 5.600000 8 8

b) 8.600000 8 8

Pointerul - utilitate (cont)

- Pentru **referirea** datelor si structurilor de date;
- Pentru manevrarea ***directa*** a memoriei;
- Pentru crearea de noi variabile in timpul executiei, prin ***alocare dinamica a memoriei.***

Transferul parametrilor

Transferul parametrilor

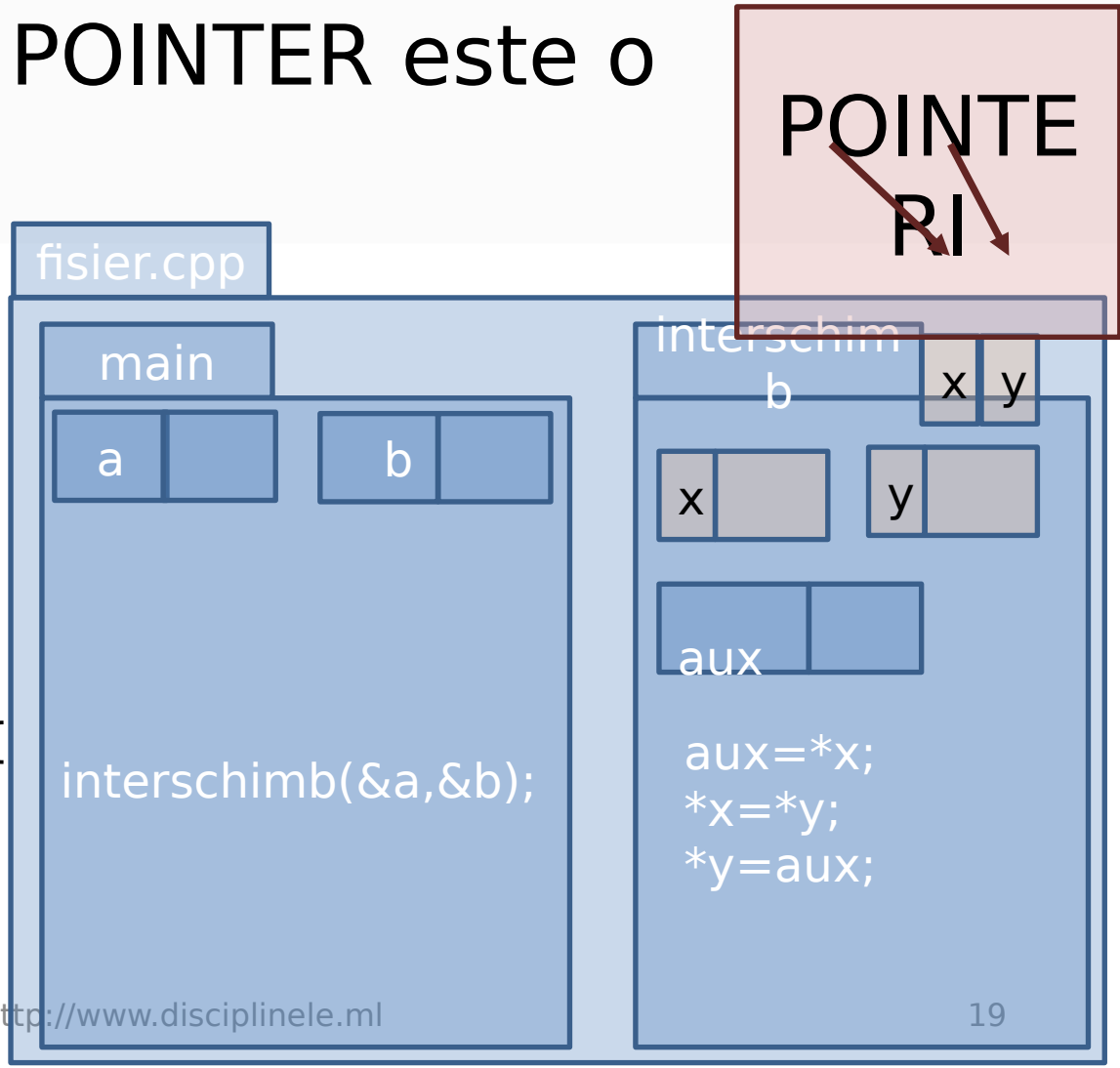
- Transferul parametrilor la functii se face **prin valoare**
  - La apel, in timpul executiei, parametrii formali sunt inlocuiti cu valorile parametrilor efectivi care le corespund.

- Daca parametrul este POINTER, valoarea sa va fi **adresa** unei *variabile*, caz in care functia reuseste sa manevreze indirect valoarea *variabilei*
  - In acest caz, functia se declara cu parametru tip pointer.

Exercitiul 4.1 – reconsiderare

- VALOAREA unui POINTER este o ADRESA!!!

```
void interschimb(int *,int *)
int main() {
  int a=5, b=7;

  printf("%d,%d",a,b);
  interschimb(&a,&b);
  printf("%d,%d",a,b);
}
void interschimb(int *x,int *y){
  int aux;
  aux=*x;
  *x=*y;
  *y=aux;
}
```



Aritmetica pointerilor

Operatii aritmetice cu pointeri

- Fie declaratiile urmatoare

```
int *p;
int i;
p = &i;
```

- Expresia **p+n** are ca valoare **valoarea lui p marita cu n*sizeof(int)**.

- **p-n** are ca valoare **p – sizeof(int)*n**
  - p++ are ca valoare valoarea lui p marita cu sizeof(int)
  - p–- are ca valoare valoarea lui p micsorata cu sizeof(int)

Exemplu

- Ce afiseaza instructiunile urmatoare

```
int i = 5, *pi=&i;
printf("%p\n", pi);
pi++;
printf("%p\n", pi);
pi++;
printf("%p\n", pi);
```

Exemplu

```
#include<stdio.h>
int main(){
    double n = 5.6, m=1.0;
    double *pn;
    pn=&n;
    *pn += 3;

    printf("1 %lf  %ld   %ld\n", n, sizeof(n), sizeof(pn));

    printf("2 p %p p+1 %p p-1 %p\n", pn, pn+1, pn-1);

    printf("3 m %lf\n", m);
    *(pn+1) = 0;

    printf("4 m %lf *(pn+1) %lf\n", m, *(pn+1));
}
```

1 observati tipul de date pentru pointer

2 observati efectul operatiilor aritmetice asupra pointerului

3 Observati valoarea lui m.

4 Observati cum este suprascris m, prin acces direct la memorie (unsafe!).

Aritmetica pointerilor

- In cazul masivelor de date (vectori, matrice, etc), **identificatorul** masivului (*numele*) este echivalent cu **adresa masivului**, sau cu **adresa primului sau element**.

int v[10]={0,4,2,7,5,9} ,i=3,j,k; // **v este de fapt asociat cu adresa unde incepe vectorul**

int *pi=v;// echivalent cu &v **sau cu** &v[0]

k=*pi; // k ia val v[0]

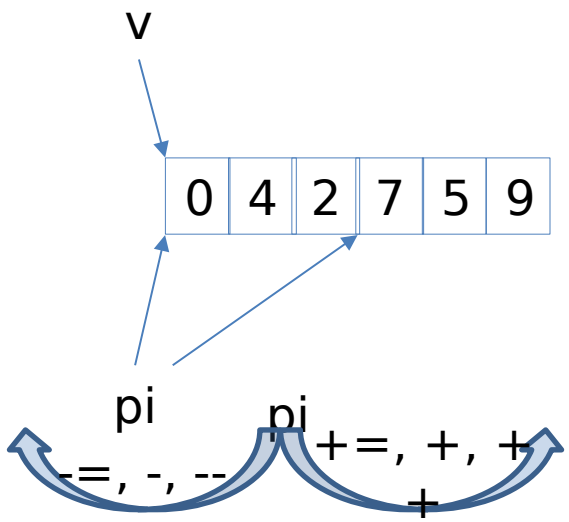
j=pi[i]; // k ia val v[i]

k=*(pi+i);// k ia val v[i]

printf("%d %d",j,k);

pi+=2;

printf("%d",*pi);



Pericol: Pierdem originea vectorul

Aritmetica pointerilor - cont

- De indata ce avem un pointer ‘legat’ de un tablou, putem face operatii aritmetice cu acel pointer
- **Aceste operatii aritmetice se traduc in deplasari in interiorul tabloului.**

- ATENTIE! O variabila de tip pointer poate primi diverse valori in timpul executiei, insa **numele unui tablou** are *tot timpul* ca valoare **adresa primului element**

- Numele unui tablou = pointer **constant**!

Operatii cu pointeri

- Doi pointeri de acelasi tip
  - pot fi **comparati** folosind operatori de relatie si egalitate
    - Constanta speciala NULL (0) este folosita in contextul comparatiilor cu == si != pentru a testa daca un pointer este nul (adica dc un pointer nu defineste o adresa valida)
  - Pot fi **scazuti**:
    - int *p=&t[2], *q = &t[5];
    - q-p are valoarea 3
    - p < q esteadevarat

Accesul direct la masive de date

Transferul parametrilor (continuare)

Parametrii tip masiv (tablou) de date pot fi transmişi la apelul unei funcții.

- Funcţia apelată lucrează **direct** cu masivul de date din funcţia apelantă

```
void test(int v[10]) {
  v[0] = 0;
}
int main(){
  int v[10] = {1, 2, 3};
  printf("v[0] este %i.\n", v[0]);
  test(v);
  printf("v[0] este %i.\n", v[0]);
}
```

Transferul parametrilor (cont)

- In C, la un astfel de apel, **nu se verifica dimensiunea** tabloului trimis ca parametru.

```
void test(int v[10]) {
  v[0] = 0;
}
int main(){
  int v[] = {1, 2, 3}; //dimensiunea este de 3 elemente
  printf("v[0] este %i.\n", v[0]);
  test(v); //apelul acesta este perfect legal
  printf("v[0] este %i.\n", v[0]);
}
```

Transferul parametrilor (cont)

- Explicație
  - Numele unei variabile de tip tablou este asociat în C cu adresa primului element din tablou.
    - int v[10];
    - Numele v este echivalent cu &v[0]

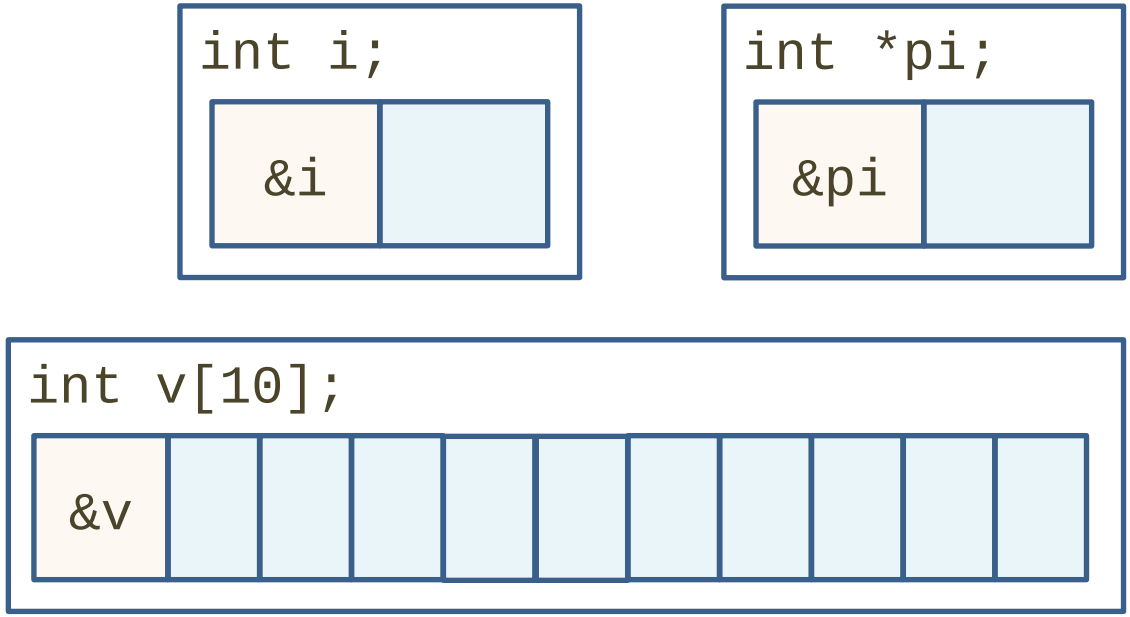
Când este trimis ca argument, se trimite așadar adresa de început a tabloului. **C nu trimite si dimensiunea tabloului**.

- Este deci OBLIGATORIU ca, la trimiterea unui tablou ca argument al unei funcții, să se trimită (aceleiași funcții) și dimensiunea tabloului, ca parametru (parametri) separati)
  - Motivație: astfel, in cadrul funcției, se va ști până la ce dimensiune este legal să fie parcurs tabloul

Exercitiul 4.5

- Specificati pentru fiecare instructiune in parte continutul casutelor albastre:

```
int i=3;
int *pi=&i;
i++;
(*pi)++;
int v[10]={0};
pi=v;//pi=&v;//pi=&v[0];
(*pi)++;
pi=&v[3];
(*pi)++;
pi+=3;
printf(“%d”,*(pi-2));
printf(“%d”,*(pi+1));
printf(“%d”,pi-v);
```



Exemplu

```
// incrementeaza primele 10 pozitii dintr-un vector
void incrementeaza(int [ ]);
// afiseaza10 elemente dintr-un vector
void afiseaza (int [ ]);
//media aritmetica a nr. de pe primele n pozitii din vector
void medie_aritmetica(int n, int t[ ]);

int main(){
  int t[10] = {1, 2, 3, 4};
  afiseaza(t); //1 2 3 4 0 0 0 0 0 0
  float medie = medie_aritmetica(5, t);
  printf("Media primelor 5 numere este %f.\n", medie);
  incrementeaza(t);
  afiseaza(t); // 2 3 4 5 1 1 1 1 1 1
  medie = medie_aritmetica(5, t);
  printf("Media primelor 5 numere este %f.\n", medie);
}
```

Exemplu (cont.)

```
float medie (int n, float v[]){
  float suma = 0;
  int i;
  for(i = 0; i < n; i++)
  suma += v[i];
  return suma/n;
}
void afiseaza (int v[]){
   for(int i=0; i<10; i++)
     printf("%d ", v[i]);
   printf("\n");
}
void incrementeaza(int v[]){
   for (int i=0; i<10; i++)
     v[i]++;
}
```

Exemplu (cont)

Modificati programul anterior asa incat sa poata lucra cu primele m elemente din vector (in loc de primele 10!), unde m este citit de la tastatura.

IDEE: pentru a putea trata pe m dinamic, este necesar a-l trimite ca parametru functiilor care depind de el!

Exercitii propuse

1) Scrieti o functie care verifica aparitia unui numar intreg printre elementele unui vector de numere intregi (0-inexistenta, 1-aparitie):aparitii(x,v,n). Imbunatatiti implementarea prin returnarea numarului de aparitii.

2) Scrieti o functie care determina minimul/maximul elementelor unui vector: int min(int v[], int n), apoi int max(int v[],int n).

3) Scrieti o functie care calculeaza si returneaza suma elementelor unui vector de numere intregi.

4) Convertiti litere mari ale unui sir de caractere in litere mici.

5) Scrieti o functie in care eliminati caracterele “albe” consecutive ale unui sir de caractere primit ca parametru.
Exemplu: “a b c   d” -> “a b c d”

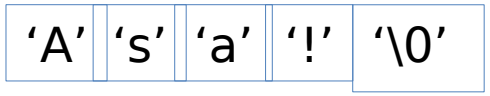
6) FOLOSITI FUNCTIILE DE LA 1 – 5 intr-un program, pentru a le demonstra functionalitatea!

ALTE EXEMPLE
DE STUDIAT!

Exercitiul 4.6

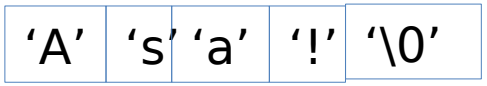
- Calculati lungimea reala a unui sir de caractere, continut intr-un vector declarat asa: char mes[256];

```
char mes[256] = “Asa!”;
 scanf(“%s”, mes); // echivalent cu gets(s);
 int i=0;
 while(mes[i]!=‘\0’)
   i++;
 printf(“Lungimea sirului este %d”,i);
```



Exercitiul 4.7

- Scrieti o functie care calculeaza lungimea REALA a unui sir continut intr-un array char mes[256]:

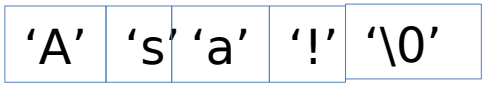



```
int lungime(char sir[ ]);
int main(){
   char mes[256];
   printf(“Lungimea sirului este %d”, lungime(mes));
   //…
}
int lungime(char sir[ ]) {
  int i=0;
  while (sir[i]!=‘\0’)
        i++;
  return i;
}
```

Exercitiul 4.8 – ver 1

- Scrieti o functie care calculeaza lungimea unui sir oarecare de caractere:

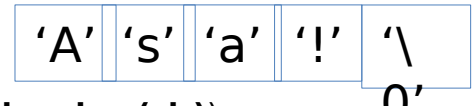
```
int lungime(char *);// int lungime(char []);
//...
char mes[256];
char sir[50];
// citim -> mes; -> sir;
printf("Lungimea lui mes este %d iar a lui sir este
   %d",lungime(mes),lungime(sir));
//...

int lungime(char * ps) {
  int lung=0;
  while (ps[lung]!='\0') lung++;
  return lung;
}
```



Exercitiul 4.8 – ver 2

- Scrieti o functie care calculeaza lungimea unui sir oarecare de caractere:

```
int lungime(char *);// int lungime(char []);
//...
char mes[256];
char sir[50];
-> mes; -> sir;
printf("Lungimea lui mes este %d iar a lui sir este %d",lungime(mes),lungime(sir));
//...
```

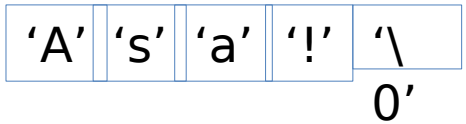



```
int lungime(char *ps) {
  int i=0;
  while (*ps!='\0') {
     ps++;
     i++;
  }
  return i;
}
```

Exercitiul 4.8 – ver 3

- Scrieti o functie care calculeaza lungimea unui sir oarecare de caractere:

```
int lungime(char *);// int lungime(char []);
//...
char mes[256];
char sir[50];
-> mes; -> sir;
printf("Lungimea lui mes este %d iar a lui sir este
   %d",lungime(mes),lungime(sir));
//...
int lungime(char *ps) {
  char *pc=ps;
  while (*ps!='\0')
      ps++;
  return ps-pc;
}
```



Gestiunea dinamica a memoriei

Pointeri de tip void *

- Un pointer declarat de tip void

  void * p;

  poate tine minte adrese de zone de memorie care pot contine date de tipuri diferite.
- Pentru a fi utilizat, pointerul void trebuie convertit explicit prin operatorul cast, a.i. sa precizeze tipul datei spre care pointeaza.

```
int x;
float y;
char c;
void *p;
p = &x;
```

```
*p=5; // eroare
*(int*)p=5; // corect
p=&y;
*(float*)p= 1.5;
p=&c;
*(char*)p='a';
```

Alocarea dinamica a memoriei

- Dimensiunea tablourilor este cunoscuta la compilare.
- Crearea de tablouri de date aloca memoria la inceputul executiei; memoria nu poate fi redimensionata pe parcursul rularii programului

- Prin alocare dinamica se permite programului sa aloce memorie **in timpul rularii** sau sa elibereze memorie alocata atunci cand aceasta nu mai este necesara

Alocarea dinamica a memoriei

- Functiile de gestionare a memoriei au prototipurile in stdlib.h
  - Functii pentru alocare dinamica a memoriei : de obicei, primesc ca parametri dimensiunea memoriei care trebuie alocata
    - malloc
    - calloc
    - realloc
    - Functiile intorc pointer la zona alocata.
    - Este posibil ca alocarea sa nu fie posibila (programul nu dispune de suficienta memorie, ceva a mers rau...). Intotdeauna pointerul returnat trebuie verificat sa nu fie NULL.
  - Functii pentru eliberare dinamica a memoriei:
    - free

malloc

- Cea mai utilizata functie de alocare a memoriei:

  void* malloc(size_t size);

  - Primeste ca parametru un numar intreg ce reprezinta dimensiunea blocului de memorie alocat
  - Functia malloc returneaza un pointer la blocul alocat de dimensiune size (daca s-a efectuat alocarea cu succes), altfel returneaza NULL.
  - Tipul returnat este void *. Acesta poate fi modificat prin conversie explicita (cast) la orice tip de pointer
- Sintaxa de utilizare
  - ptr=(cast-type*)malloc(byte-size)
- Exemplu
  - ptr=(**int***)malloc(100***sizeof**(**int**));

  Instructiunea aloca spatiu pentru 100 de elemente de tip int (deci 400 de bytes) si returneaza pointer la inceputul zonei alocate (la primul byte, din primul int).

free

- Functia free elibereaza blocul alocat anterior cu malloc, avand pointerul transmis ca parametru
  - void free(void * block);
- Sintaxa de utilizare
  - free(ptr)
- Exemplu

```
int * ptr=(int*)malloc(100*sizeof(int));
……
free(ptr);
```

Alocarea dinamica a memoriei

- /* Un program care citeste un sir de n numere intregi, n citit de la tastatura. Atentie. Memoria necesara va fi alocata dinamic, in momentul executiei.*/

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    int n, * p;
    printf("n=");
    scanf("%d", &n);
    p=(int*)malloc(n*sizeof(int));
    if(p==NULL) {
        printf(“Nu s-a alocat memorie”);
        system(“pause”);
        return 1;
    }
```

```
for(int i=0; i<n; i++) {
   printf("Introduceti nr. la pozitia %d:", i);
   scanf("%d", p+i); //echiv. cu &p[i]
    }

for(int i=0; i<n; i++){
     printf("Element la pozitia %d: %d.\n", i, *(p+i)); // echiv cu p[i]
    }
// la final, eliberam memoria alocata
    free(p);
    system("pause");
    return 0;
}
```

calloc

- void * calloc (size_t num, size_t size);
  - Aloca un bloc de memorie de *num* elemente de dimensiune *size;* memoria este initializata cu zero; elementele sunt alocate contiguu in memorie
  - Returneaza pointer la zona alocata (NULL daca nu s-a efectuat cu succes alocarea)
  - http://www.cplusplus.com/reference/clibrary/cstdlib/calloc/
- Sintaxa de utilizare
  - ptr=(cast-type*)calloc(n,element-size);
- Exemplu
  - ptr=(**float***)calloc(25,**sizeof**(**float**));

  Instructiunea aloca spatiu pentru 25 de elemente contigue de tip float(deci 100 de bytes) si returneaza pointer la inceputul zonei alocate (la primul byte, din primul int). Toti bitii sunt setati la 0 (deci numerele float sunt 0).

realloc

- void * realloc (void * ptr, size_t size);
  - In cazul in care memoria alocata anterior este prea mare, sau prea mica, se poate modifica dimensiunea memoriei referita de *ptr.*
- Sintaxa de utilizare

  ptr=realloc(ptr,newsize);
- Exemplu de utilizare

  float * ptr=(**float***)calloc(25,**sizeof**(**float**));

  ptr = realloc(ptr, 50*sizeof(float))
- Daca ptr este null, functia aloca spatiu (precum malloc)

Exercitii

Scrieti un program care citeste n numere de la tastatura (n citit de asemenea de la tastatura) si calculeaza media numerelor citite si maximul dintre acestea, apoi le afiseaza in ordine crescatoare. Se va folosi alocare dinamica pentru stocarea numerelor citite.

Exercitii

- /* Scrieti o functie care face acelasi lucru cu functia **strcmp** – compara doua siruri de caractere dpdv lexicografic si returneaza -1 daca primul sir este mai mic decat al doilea, 0, daca sirurile sunt egale si 1 daca primul sir este mai mare.*/

```
int my_strcmp( char *s1,
char *s2){
   int i;
   for(i=0;
     s1[i]!=0 || s2[i] != 0; i+
+)
      if(s1[i]<s2[i])
        return -1;
      else
        if(s1[i]>s2[i])
          return 1;
   return 0;
  }
```

Parcurg cat timp nu s-a ajuns la sfarsitul unuia dintre siruri