# Curs 1:

# Introducere în rezolvarea algoritmică a problemelor

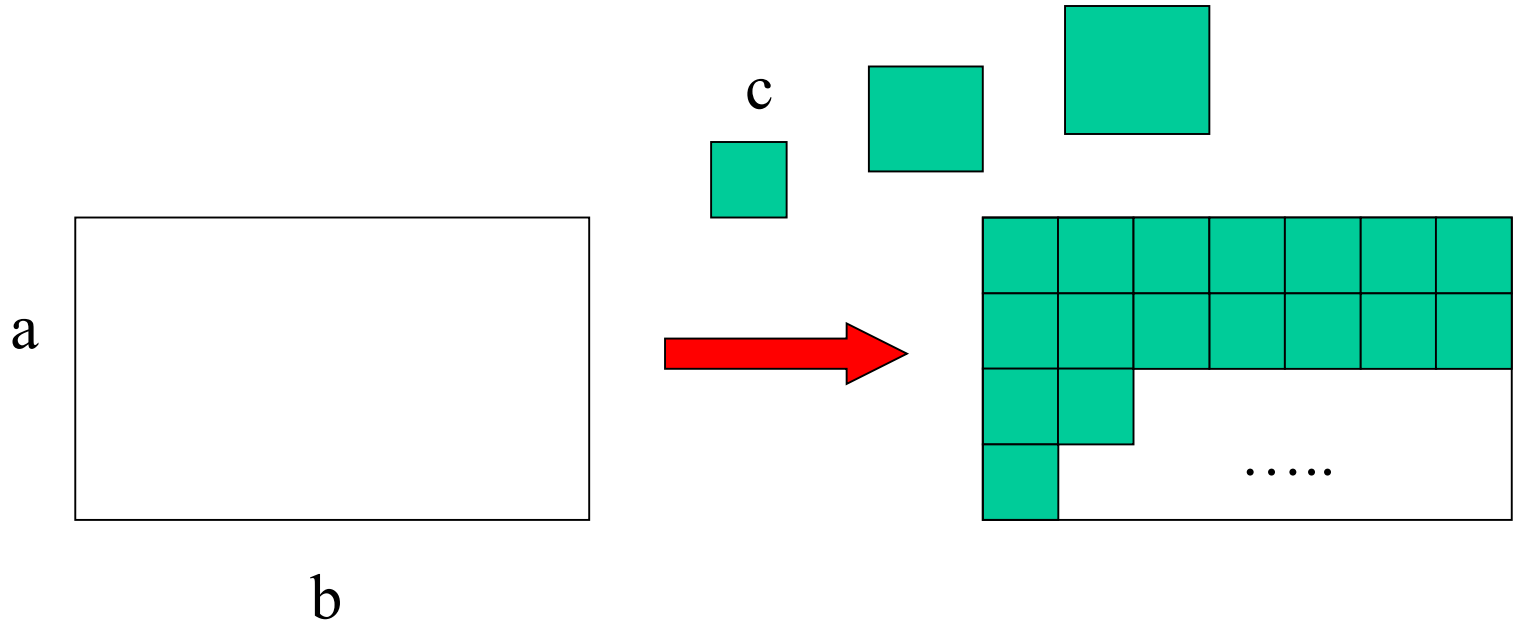
# Cuprins

- Rezolvarea problemelor
- Ce este un algoritm ?
- Ce proprietăți ar trebui să aibă un algoritm ?
- Cum pot fi descriși algoritmii ?
- Ce tipuri de date vor fi utilizate ?
- Cum pot fi specificate prelucrările dintr-un algoritm ?

# Rezolvarea problemelor

Exemplu:

Considerăm o suprafață dreptunghiulară de laturi **a** respectiv **b** (valori naturale) care trebuie acoperită în întregime cu pătrate având latura **c**. Să se determine valoarea lui **c** astfel încât numărul de pătrate utilizate să fie cât mai mic.



# Rezolvarea problemelor

Exemplu:

Se cunosc valorile naturale a și b (lungimile laturilor dreptunghiului). Se caută o valoare c care are următoarele proprietăți:

- c divide pe a și pe b (c este divizor comun al lui a și al lui b)
- c este mai mare decât orice alt divizor al lui a și b

Domeniul problemei: mulțimea numerelor naturale (a și b reprezintă datele de intrare, c reprezintă rezultatul)

Enunțul problemei (relația dintre datele de intrare și rezultat): **c** este cel mai mare divizor comun al lui **a** și **b**

# Rezolvarea problemelor

Problema = set de întrebări referitoare la anumite entități care reprezintă domeniul problemei

Enunțul problemei = descrierea proprietăților entităților și a relației dintre datele de intrare și soluția problemei

In mod uzual se specifică: “ce se dă” (input) și “ce se cere” (output)

Metoda de rezolvare = procedeu de construire a soluției pornind de la datele de intrare

# Rezolvarea problemelor

Observație:

- Problema determinării cmmdc face parte din clasa celor care calculează valoarea unei funcții (in acest caz se asociază unei perechi de numere naturale valoarea celui mai mare divizor comun).

- Un alt tip de probleme sunt cele care cer să se verifice dacă datele de intrare satisfac o anumită proprietate. Acestea sunt denumite probleme de decizie.

  Exemplu: să se verifice dacă un număr natural este prim sau nu

In ambele cazuri soluția poate fi obținută folosind un calculator doar dacă există o metodă care să furnizeze rezultatul după un număr finit de prelucrări care pot fi executate de către calculator … o astfel de metodă este un algoritm

# Cuprins

- Rezolvarea problemelor
- Ce este un algoritm ?
- Ce proprietăți ar trebui să aibă un algoritm ?
- Cum pot fi descriși algoritmii ?
- Ce tipuri de date vor fi utilizate ?
- Cum pot fi specificate prelucrările dintr-un algoritm ?

# Ce este un algoritm ?

Există diferite definiții …

Algoritm = o descriere pas cu pas a metodei de rezolvare a unei probleme

Algoritm = o succesiune finită de operații care aplicate datelor de intrare ale unei probleme conduc la soluție

Algoritm = rețetă de rezolvare a unei probleme

…

# Care este originea cuvântului ?

al-Khowarizmi - matematician persan (aprox. 790-840)


- A fost printre primii ce a folosit cifra 0
- A scris prima carte de algebră (numele acestei discipline provine de la același matematician)

# Exemple

Algoritmi în viața de zi cu zi:

- Utilizarea unui bancomat, automat pentru cafea etc.

Algoritmi specifici matematicii:

- Algoritmul lui Euclid (este considerat primul algoritm)
  - Determinarea celui mai mare divizor comun a două numere
- Algoritmul lui Eratostene
  - Generarea numerelor prime mai mici decât o valoare dată
- Algoritmul lui Horner
  - Calculul valorii unui polinom sau a restului împărţirii la un binom de forma (X-a)


Euclid
(cca. 325 -265 i.C.)

# De la problemă la algoritm

**Problema:**

- Datele problemei
  - a, b - nr.naturale
- Cerința:
  - determină cmmdc(a,b)

Metoda de rezolvare

- împarte a la b și reține restul
- împarte b la rest și reține noul rest
- continuă împărțirile până se ajunge la un rest nul
- ultimul rest nenul reprezintă rezultatul

# De la problema la algoritm

**Problema:**

- Datele problemei
  - a, b - nr.naturale
- Cerinta:
  - determina cmmdc(a,b)
- Metoda de rezolvare
  - împarte a la b și reține restul
  - împarte b la rest și reține noul rest
  - continuă impărțirile până se ajunge la un rest nul
  - ultimul rest nenul reprezintă rezultatul

**Algoritm:**

- Variabile = obiecte abstracte ce corespund datelor problemei + datelor de lucru
  - deîmpărțit, împărțitor, rest
- Secvența de prelucrări
  1. Atribuie deimpărțitului valoarea lui a și împărțitorului valoarea lui b
  2. Calculează restul împărțirii deîmpărțitului la împărțitor
  3. Atribuie deîmpărțitului valoarea împărțitorului și împărțitorului valoarea restului anterior
  4. Daca restul e nenul reia de la etapa 2

# De la algoritm la program

**Algoritm:**

- Variabile = obiecte abstracte ce corespund datelor problemei
  - deîmpărțit, împărțitor, rest
- Secvența de prelucrări
  1. Atribuie deîmpărțitului valoarea lui a și împărțitorului valoarea lui b
  2. Calculează restul împărțirii deîmpărțitului la împărțitor
  3. Atribuie deîmpărțitului valoarea împărțitorului și împărțitorului valoarea restului anterior
  4. Daca restul e nenul reia de la etapa 2

**Program:**

- Variabile = obiecte abstracte ce corespund datelor problemei
  - Fiecare variabilă are asociată o zonă în memoria calculatorului
- Secvența de instrucțiuni
  - Fiecare instrucțiune corespunde unei prelucrări elementare care poate fi executată de către calculator

**Model (extrem de) simplificat**


Intrare/iesire

Unitate memorie

Unitate calcul

# Cuprins

- Rezolvarea problemelor
- Ce este un algoritm ?
- Ce proprietăți ar trebui să aibă un algoritm ?
- Cum pot fi descriși algoritmii ?
- Ce tipuri de date vor fi utilizate ?
- Cum pot fi specificate prelucrările dintr-un algoritm ?

# Ce proprietăți ar trebui să aibă un algoritm ?

Un algoritm trebuie să fie:

- General = este aplicabil unei întregi clase de probleme nu doar unor cazuri particulare
- Corect (inclusiv finit)
- Eficient

# Generalitate

Un algoritm trebuie să funcționeze corect pentru toate instanțele de date de intrare nu doar pentru cazuri particulare

Exemplu:

Să considerăm problema ordonării (sortării) crescătoare a unui șir de valori numerice.

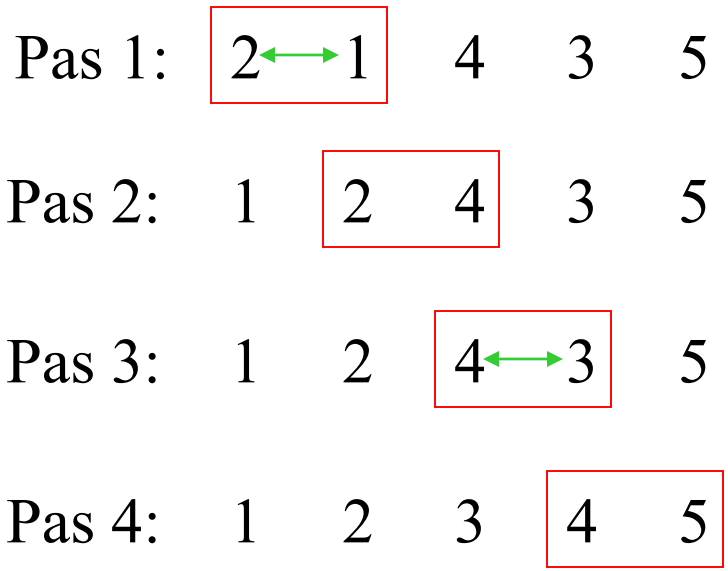
De exemplu:

(2,1,4,3,5) ⟶ (1,2,3,4,5)

date intrare — rezultat

# Generalitate

## Metoda:




## Descriere:

-Compară primele două elemente; dacă nu sunt în ordinea dorită se interschimbă

-Compară al doilea cu al treilea și aplică aceeași strategie

…..

- Continuă procesul până la ultimele două elemente din secvență

Secvența a fost ordonată (după 4 comparații)

# Generalitate

- Este acest algoritm suficient de general ? Asigură ordonarea crescătoare a oricărui șir de valori ?
- Răspuns: NU

  Contraexemplu:

3 2 1 4 5
2 3 1 4 5
2 1 3 4 5
2 1 3 4 5

In acest caz metoda nu funcționează deci nu poate fi considerată un algoritm general de sortare. Pentru a realiza sortarea completă e necesară reluarea procesului de parcurgere a secvenței:

2 1 3 4 5 -> 1 2 3 4 5 -> 1 2 3 4 5 -> 1 2 3 4 5 -> 1 2 3 4 5

# Intrebare

Parcurgere 1:

3 2 1 4 5
2 3 1 4 5
2 1 3 4 5
2 1 3 4 5

Parcurgere 2:

2 1 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5

Care este numărul de parcurgeri ale unui șir arbitrar ce conține n valori care garantează faptul că șirul va fi ordonat?

Răspuns: n-1

Remarcă: această variantă de algoritm de sortare este printre cele mai puțin eficiente

# Ce proprietăți ar trebui să aibă un algoritm ?

Un algoritm trebuie să fie:

- General
- Corect (inclusiv finit) = algoritmul conduce la rezultat după un numări finit de prelucrări
- Eficient

# Finitudine

- Un algoritm trebuie să se termine după un număr finit de prelucrări

Exemplu

Pas1: Asignează 1 lui x;

Pas2: Adaugă 2 la x;

Pas3: Dacă x=10 atunci STOP;

altfel se reia de la Pasul 2

Cum lucrează acest algoritm ?

# Finitudine

Cum lucrează algoritmul și ce produce:

Pas1: Asignează 1 lui x; x=1

Pas2: Adaugă 2 la x; x=3 x=5 x=7 x=9 x=11

Pas3: Dacă x=10 atunci STOP;

altfel afișează x; se reia de la Pasul 2

Ce putem spune despre acest algoritm ?

Afișează numere impare dar nu se oprește niciodată !

# Finitudine

Algoritmul care generează numerele impare mai mici decat 10:

Pas1: Asignează 1 lui x;
Pas2: Adaugă 2 la x;
Pas3: Dacă x>=10 atunci STOP;
altfel afișează x; se reia de la Pasul 2

# Ce proprietăți ar trebui să aibă un algoritm ?

Un algoritm trebuie să fie:

- General
- Corect (inclusiv finit)
- Eficient = necesită un volum rezonabil de resurse de calcul (spatiu de stocare, timp de executie)

# Eficiență

Finitudinea nu e suficientă dacă timpul necesar obținerii unui rezultat este prea mare

Exemplu:

- Să presupunem că avem acces la o listă care conține coduri constituite din 10 cifre asociate unor nume
- Lista este ordonată crescător după cod
- Putem parcurge lista secvențial (începând cu primul elemente) sau putem exploata faptul că este ordonată

Ne intereseaza să identificăm o persoană despre care știm că are asociat un cod constituit din 10 cifre distincte însă nu știm care este codul și nici numele – dar presupunem ca dacă am vedea numele ne-am aminti

Cum putem proceda?

# Eficiență

Prima abordare:

Pas 1: generează toate codurile constituite din 10 cifre distincte

Pas 2: pentru fiecare cod generat se caută în listă și se analizează numele

A doua abordare:

se parcurge lista secvențial si pentru fiecare cod se verifică dacă este constituit din cifre distincte sau nu, iar în caz afirmativ se analizează numele

**Care variantă este mai bună (în raport cu numărul de operații efectuate – numărul de nume analizate)?**

# Eficiență

Prima abordare:

Pas 1: generează toate codurile constituite din 10 cifre distincte

Pas 2: pentru fiecare cod generat se analizează numele corespunzător din listă

Notăm cu:

- m = numărul de cifre ale codului (ex: m=10)
- n = numărul de elemente din lista

O estimare a numărului de operații elementare (comparații):

- Numărul de coduri posibile: m!
- Numărul de elemente din lista care sunt analizate: $\log_2 n$ (se folosește metoda căutării binare – vezi curs 7)
- Numărul de cifre comparate pentru fiecare cod: m

$m!\ m \log_2 n$

# Eficiență

A doua abordare:

se parcurge lista secvențial și pentru fiecare cod se verifică dacă este constituit din cifre distincte sau nu

Estimarea numărului de comparații:

– Parcurgerea celor n înregistrări și verificarea dacă numărul conține doar cifre distincte - cel mult $m^2$ comparații

**$n\ m^2$** (în varianta de tip forță brută)

sau

**$nm$** (bazat pe tabel de frecvențe a cifrelor)

# Eficiență

Care abordare este mai bună ?

Prima abordare

**$m! \; m \log_2 n$**

A doua abordare

**$n \; m$**

Depinde de valorile lui m și n

Exemplu: m = 10

n = 306.462

(populația Timișoarei la recensământul din 2012)

Aproximativ

**$6.6 * 10^8$** **$3*10^6$** **operații**

# Eficiență

Care abordare este mai bună ?

| Prima abordare | A doua abordare |
| --- | --- |
| **$m!\ m \log_2 n$** | **$n\ m$** |

Exemplu: m = 10

n = 306.462

(populația Timișoarei la recensământul din 2011)

| Aproximativ | |
| --- | --- |
| **$6.6 * 10^8$** | **$3*10^6$ operații** |

**Dar dacă m=20** (cod constituit din litere)

| | |
| --- | --- |
| **$8.8*10^{20}$** | **$6*10^6$** |

# Eficiență

O scurtă analiză (imprecisă):

$8.8*10^{20}$ operații înseamnă aproximativ $7*10^{7}$ secunde pe un calculator cu o putere de procesare de 11.7 Tflops

($11.7*10^{12}$ operații/secundă)

adică cca 20000 ore = 870 zile = 2.3 ani

Ineficiența algoritmului nu poate fi întotdeauna compensată prin sporirea resurselor de calcul

Obs: cel mai puternic calculator in 2019: **Summit** (IBM) - 148.6 **petaflops** – $148.6*10^{15}$ operații/secundă (https://www.top500.org/lists/2019/06)

# Cuprins

- Rezolvarea problemelor
- Ce este un algoritm ?
- Ce proprietăți ar trebui să aibă un algoritm ?
- Cum pot fi descriși algoritmii ?
- Ce tipuri de date vor fi utilizate ?
- Cum pot fi specificate prelucrările dintr-un algoritm ?

# Cum pot fi descriși algoritmii ?

Metodele de rezolvare a problemelor sunt de regulă descrise într-un limbaj matematic

Limbajul matematic nu este întotdeauna adecvat întrucât:

- Operații considerate elementare din punct de vedere matematic nu corespund unor prelucrări elementare când sunt executate pe un calculator.

Exemple: calculul unei sume, evaluarea unui polinom etc

Descriere matematică

$$\sum_{i=1}^{n} i = 1 + 2 + \ldots + n$$

Descriere algoritmică

?

# Cum pot fi descrișI algoritmii?

Există cel puțin următoarele modalități:

- Scheme logice (diagrame):
  - Descrieri grafice ale fluxului de prelucrări din algoritm
  - Sunt destul de rar utilizate la ora actuală
  - Totuși pot fi utile în descrierea structurii generale a unei aplicații

- Pseudocod:
  - Limbaj artificial bazat pe
    - vocabular (set de cuvinte cheie)
    - sintaxă (set de reguli de construire a frazelor limbajului)
    - semantică (semnificația construcțiilor din limbaj)
  - Nu e la fel de restrictiv ca un limbaj de programare

# Cum pot fi descrișI algoritmii?

Există cel puțin următoarele modalități:

- Scheme logice (diagrame)
- Pseudocod:
- Limbaj de programare
  - Limbaj artificial în care pot fi descrișI algoritmii pentru a putea fi executați de către un calculator
  - Exemple: C/C++, Python, Java …

# De ce i se spune pseudocod ?

Pentru că …

- Este oarecum similar unui limbaj de programare (cod)
- Dar nu este la fel de riguros ca un limbaj de programare (pseudo)

Frazele pseudocodului sunt de regulă:

- Instrucțiuni (utilizate pentru a descrie pașii de prelucrare)
- Declarații (utilizate pentru a specifica datele)

# Ce tipuri de date pot fi utilizate ?

Data = entitate purtătoare de informație
= container care conține o valoare

Caracteristici:

- nume
- valoare
  - constantă (aceeași valoare pe parcursul execuției algoritmului)
  - variabilă (valoarea se schimbă pe parcursul execuției algoritmului)
- tip
  - simplu (ex: numere, caractere, valori de adevăr etc.)
  - agregat / structurat (ex: tablouri, articole etc.)

# Cum pot fi specificate?

- Date simple:
  - Intregi int <nume variabila>
  - Reale float <nume variabila>
  - Logice bool <nume variabila>
  - Caractere char <nume variabila>

Obs. Există limbaje de programare (ex: Python) care nu necesită declararea explicită a datelor tipul variabilelor fiind stabilit în mod dinamic în momentul asignării unor valori

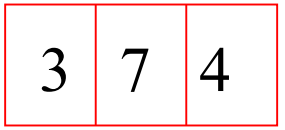
# Ce tipuri de date pot fi utilizate ?

Date structurate: tablouri

Tablourile sunt utilizate pentru a reprezenta:


- Mulțimi (obs: {3,7,4}={3,4,7})
  - Ordinea elementelor nu are importanță

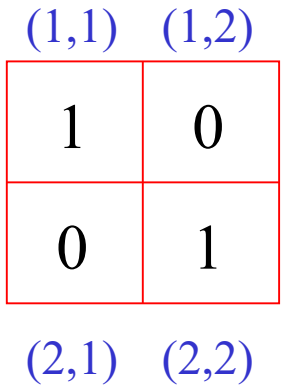
- Secvențe (obs: (3,7,4) este diferit de (3,4,7))
  - Ordinea elementelor are importanță




Index: 1 2 3

- Matrici
  - Tablouri bidimensionale (multidimensionale)

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$



# Cum pot fi specificate datele ?

Tablouri

Unidimensionale

<tip element> <nume>[n1..n2]

(ex: real x[1..n])

Bidimensionale

<tip element> <nume>[ m1..m2, n1..n2]

(ex: int A[1..m,1..n])

# Cum pot fi specificate datele ?

Specificarea elementelor tablourilor:

- Unidimensionale

  x[i] - i este indicele elementului

- Bidimensionale

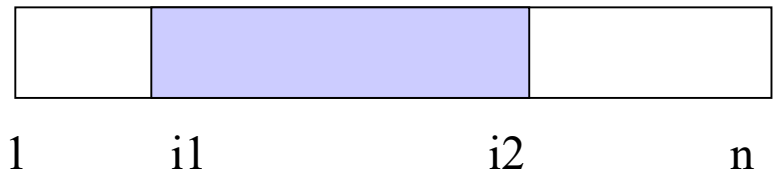
  A[i,j] - i este indice de linie, j este indice de coloană

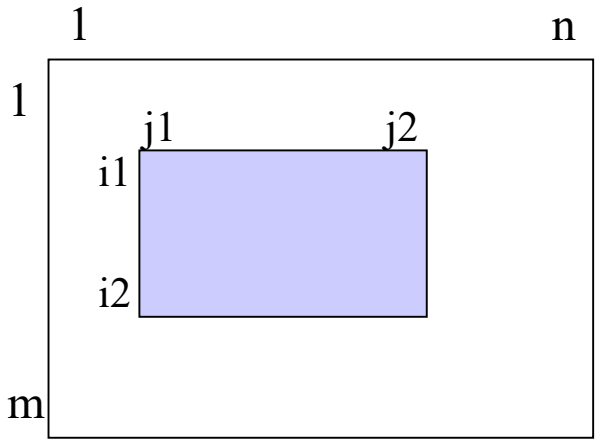
# Cum pot fi specificate datele ?

Specificarea subtablourilor

- Subtablou= porțiune contiguă a unui tablou

  - Unidimensional: x[i1..i2] (1<=i1<i2<=n)
  - Bidimensional: A[i1..i2, j1..j2]
    (1<=i1<i2<=m, 1<=j1<j2<=n)



# Alte date structurate

Exemplu: set de informatii despre studenți:
(numeStudent, seria, grupa, subgrupa)

Tip de date: articol = ansamblu de câmpuri
(câmpurile pot fi de diferite tipuri)

Referirea unui câmp:
<nume data>.<nume câmp>

Exemplu: student.seria

Obs: Articolele aferente mai multor studenți pot fi grupate în mai multe moduri

# Alte date structurate

Exemplu: set de informatii despre studenți:
(numeStudent, seria, grupa, subgrupa)

Colecție: set de articole între care nu e specificată nici o relație

(Popescu, 2,4,2)

(Avramescu, 1,1,2)

(Ionescu, 1,2,2)

(Georgescu, 2,3,1)

(Florescu, 1,2,1)

(Costescu, 2,1,2)

# Alte date structurate

Exemplu: set de informatii despre studenți:
(numeStudent, seria, grupa, subgrupa)

Lista: elementele setului sunt „aranjate” pe baza unei relații de ordine

(Popescu, 2,3,2)

(Avramescu, 1,1,2)

(Georgescu, 2,3,1)

(Ionescu, 1,2,2)

(Costescu, 2,1,2)

(Florescu, 1,2,1)

Ordine arbitrară

(Avramescu, 1,1,2)

(Costescu, 2,1,2)

(Florescu, 1,2,1)

(Georgescu, 2,3,1)

(Ionescu, 1,2,2)

(Popescu, 2,3,2)

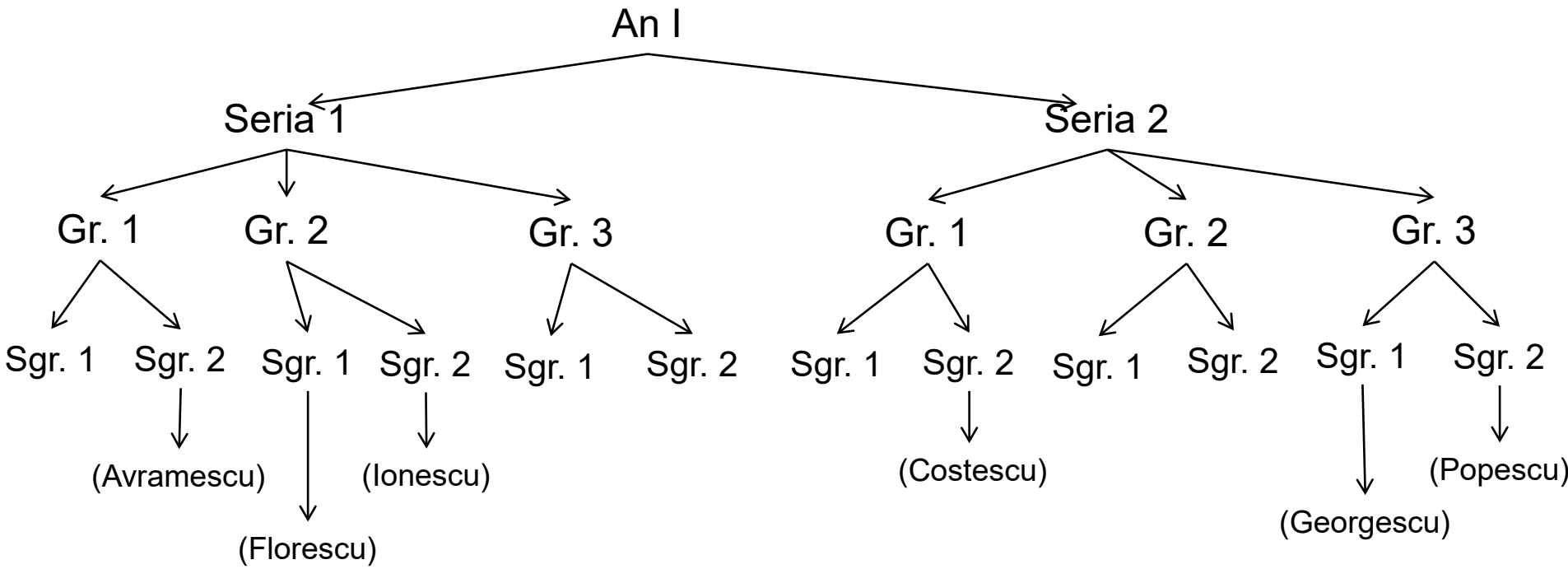
Ordine alfabetică

# Date structurate

Exemplu: set de informatii despre studenți:

(numeStudent, seria, grupa, subgrupa)

Arbore: elementele setului sunt „aranjate” într-o structură ierarhică indusă de modul de organizare pe serii/ grupe/ subgrupe

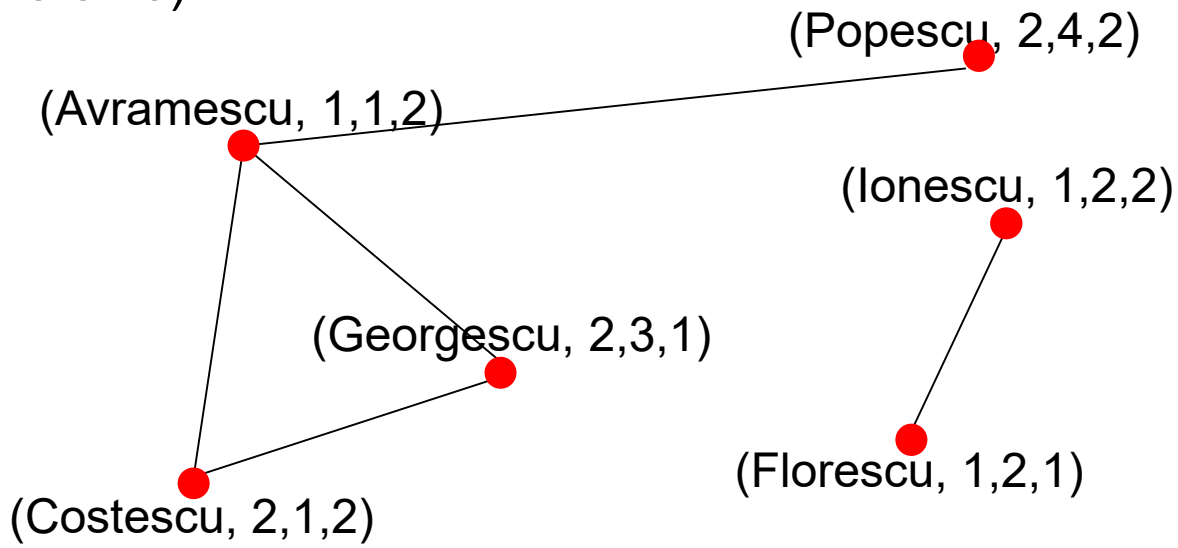


# Date structurate

Exemplu: set de informatii despre studenți:

(numeStudent, seria, grupa, subgrupa)

Graf: există o relație care „leagă” elementele setului (relația de prietenie)



# Cuprins

- Rezolvarea problemelor
- Ce este un algoritm ?
- Ce proprietăți ar trebui să aibă un algoritm ?
- Cum pot fi descriși algoritmii ?
- Ce tipuri de date vor fi utilizate ?
- Cum pot fi specificate prelucrările dintr-un algoritm ?

# Cum pot fi specificate prelucrările dintr-un algoritm ?

Instrucțiune

= acțiune (operație) executată de către un algoritm

Tipuri de instrucțiuni:

- Simple
  - Atribuire (atribuie o valoare unei variabile)
  - Control (specifică care este următorul pas care trebuie executat)
  - Transfer (preia date de intrare; afișează rezultate)
- Structurate ….

# Atribuire

- Scop: atribuie o valoare unei variabile
- Descriere:

  v← <expresie> sau v:=<expresie> sau v=<expresie>

- Expresie = construcție sintactică (= succesiune de simboluri care respectă niște reguli) utilizată pentru a descrie un calcul

Este constituită din:

- Operanzi: variabile, valori constante
- Operatori: aritmetici
  relaționali
  logici

# Operatori

- Aritmetici:
  + (adunare), - (scădere), *(înmulțire),
  / (împărțire), ^ sau ** (ridicare la putere),
  DIV sau / sau // (câtul împărțirii întregi),
  MOD sau % (restul împărțirii întregi)

- Relaționali:
  == (egal), != (diferit),
  < (strict mai mic), <= (mai mic sau egal),
  > (strict mai mare), >= (mai mare sau egal)

- Logici:
  or sau | (disjuncție), and sau & (conjuncție),
  not (negație)

Obs:

- operatorii marcați cu roșu pot fi folosiți și în Python

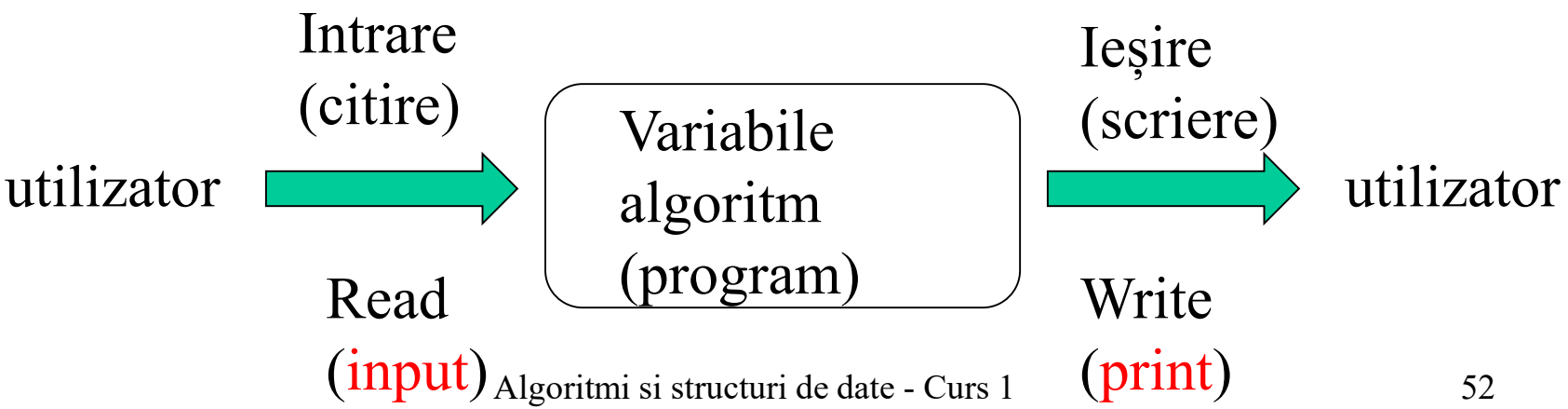
- Cei marcați cu albastru pot fi folosiți doar in pseudocod

# Intrare/ieșire

- Scop:
  - Preia date de intrare
  - Furnizează rezultate
- Descriere:

input v1,v2,…

print e1,e2,…



# Intrare/ieșire

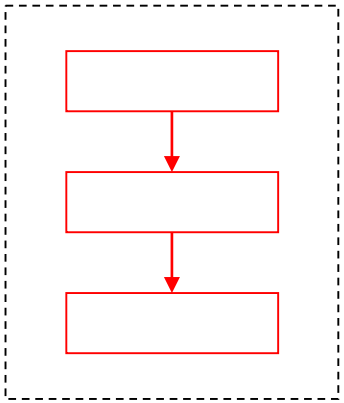
- Exemplu (Python)

```
# Calculul ariei si perimetrului unui dreptunghi
a=input("Lungime dreptunghi=")  # preluare date de intrare
b=input("Largime dreptunghi=")
aria=a*b                        # calcul arie
perimetru=2*(a+b)               # calcul perimetru
print "Arie=", aria             # afisare rezultate
print "Perimetru=", perimetru
```

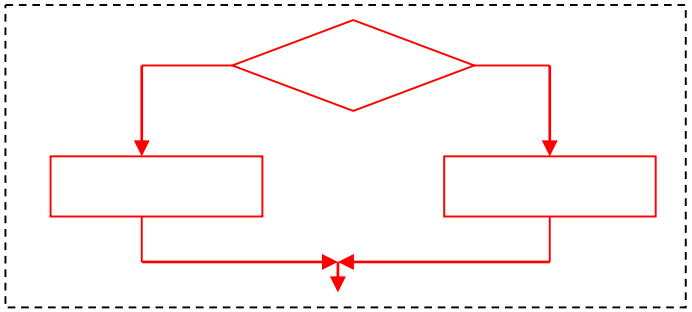
**Obs:** sintaxa pentru **`print`** depinde de versiunea de Python (v.2.7: fără paranteze, v.3.x: cu paranteze)

# Structuri de prelucrare

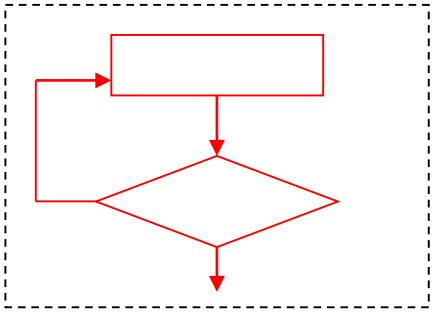
- Secvența de instrucțiuni



- Instrucțiune de decizie (condițională)

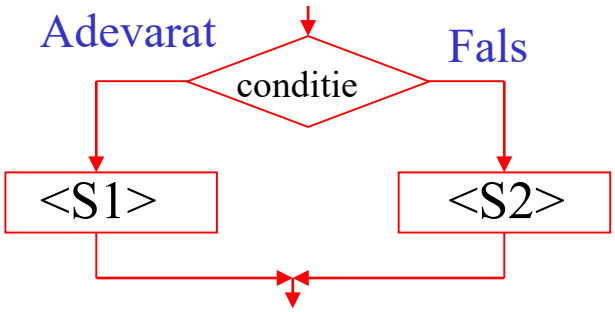


- Instrucțiune de ciclare (repetitivă)



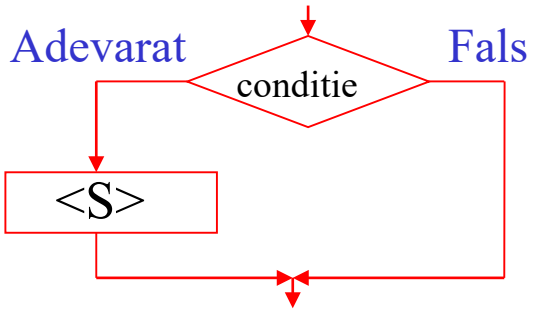
# Instrucțiune de decizie

- Scop: permite alegerea între două sau mai multe variante de prelucrare în funcție de realizarea/ nerealizarea unei (unor) condiții




- Varianta generală (pseudocod):

```
if <condiție> then  <S1>
                else  <S2>
endif
```

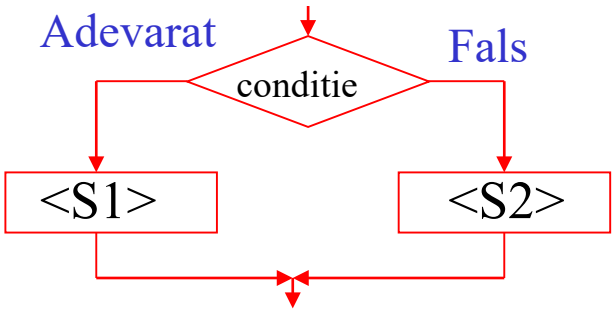


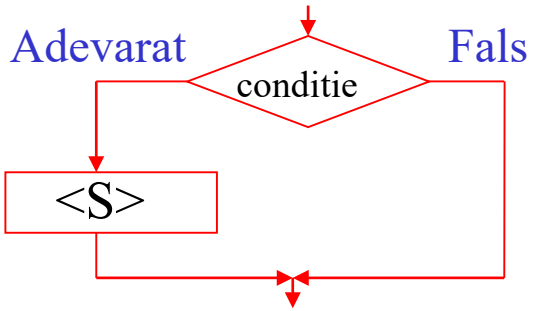

- Varianta simplificată (pseudocod):

```
if <condiție> then <S>
endif
```

# Instructiune de decizie

- Scop: permite alegerea între două sau mai multe variante de prelucrare în funcție de realizarea/ nerealizarea unei (unor) condiții




- Varianta generală (Python):

```
if <condiție>:
    <S1>
else:
    <S2>
```

- Varianta simplificată (Python):

```
if <condiție>:
    <S>
```

# Instructiune de decizie

- Exemplu: verificare dacă un număr este par sau nu

```
n=input("n=")
if n%2==0:
    print "Numar par"
else:
    print "Numar impar"
```

- Exemplu: functia signum

$$S(x) = \begin{cases} -1, & x < 0 \\ 0, & x = 0 \\ 1, & x > 0 \end{cases}$$

```
x=input("x=")
if x>0:
    s=1
elif x<0:
    s=-1
else:
    s=0
print "sgn(",x,")=",s
```

# Instrucțiuni de ciclare

- Scop: permit repetarea unei prelucrări
- Exemplu: calculul sumei

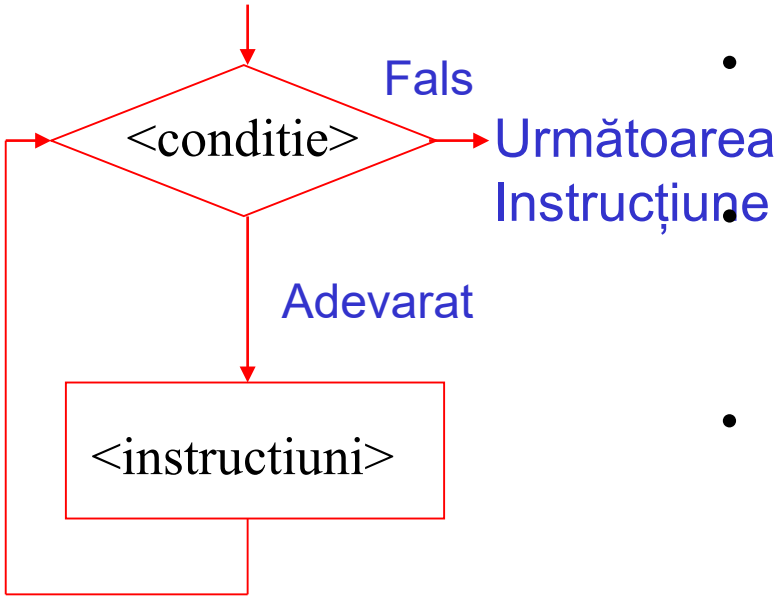
  S= 1+2+…+i+…+n
- Un ciclu este caracterizat prin:
  - Pasul de prelucrare care trebuie repetat

    (ex: adunarea următoarei valori la valoarea curentă a sumei)
  - O condiție de oprire (continuare) a prelucrării repetitive

    (ex: s-au adunat deja toate valorile)

- Depinzând de momentul în care condiția de continuare/oprire este analizată există:
  - Cicluri condiționate anterior (WHILE)
  - Cicluri condiționate posterior (REPEAT)

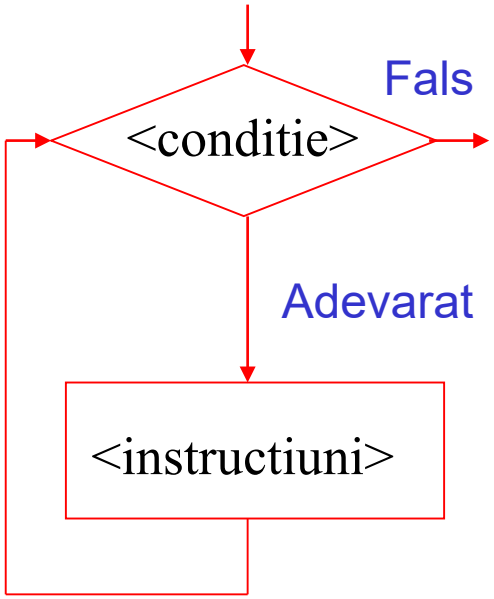
# WHILE




```
while  <condiție> do
       <instrucțiuni>
endwhile
```

- Se analizează condiția de continuare
- Dacă este adevarată se execută instrucțiunea din corpul ciclului după care se evaluează din nou condiția
- Când condiția devine falsă se trece la următoarea prelucrare din algoritm
- Dacă condiția nu devine niciodată falsă ciclul este infinit
- Dacă condiția este falsă de la început atunci corpul ciclului nu este executat niciodată

# WHILE - exemplu

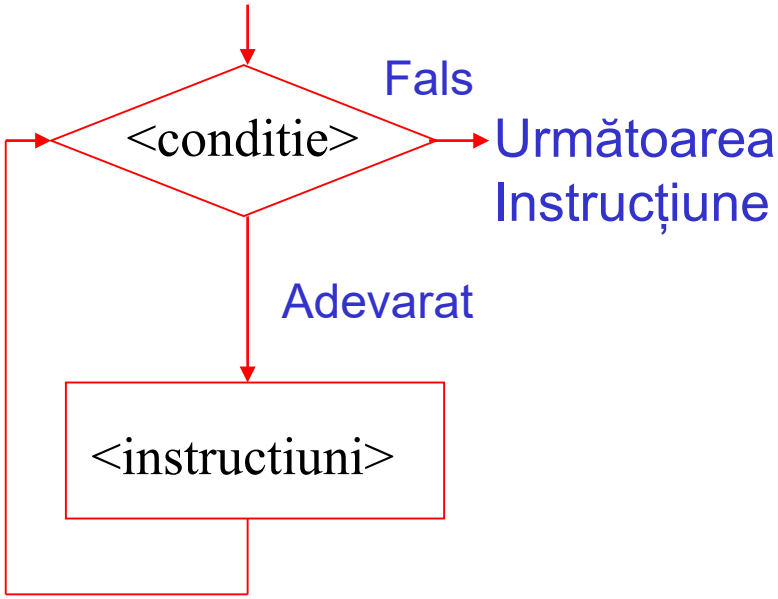



$$\sum_{i=1}^{n} i = 1 + 2 + \ldots + n$$

```
while  <conditie> do
        <instructiuni>
endwhile
```

```
S = 0    // pregătește variabila în
          //care se va colecta rezultatul
i = 1    // inițializează indicele
          // termenului de adăugat (termenul
          // coincide cu indicele)
while i<=n do
   S = S+i  // adaugă termenul la S
   i = i+1    // pregătește următorul
               //termen
endwhile
```

# WHILE - Python



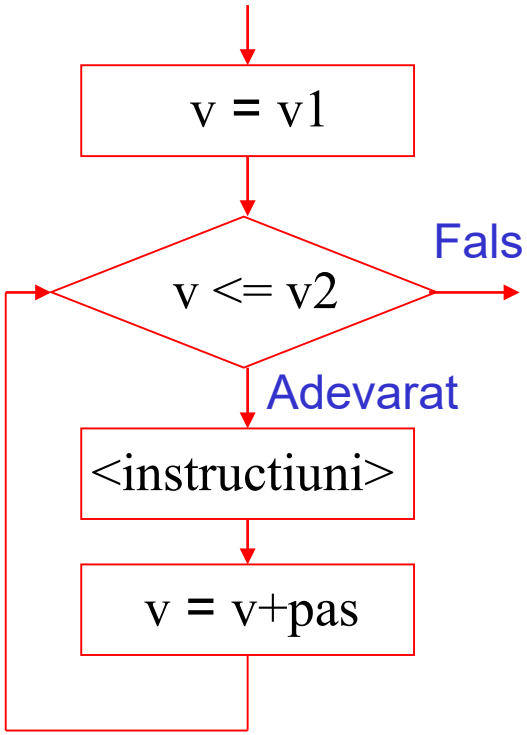

```
while <conditie>:
   <instructiuni>
```

while <conditie> do
<instructiuni>
endwhile

Exemplu:

```
# Calculul unei sume
n=input("n=")
s=0
i=1
while (i<=n):
    s=s+i
    i=i+1
print "s=",s
```

# FOR




Urmatoarea instructiune

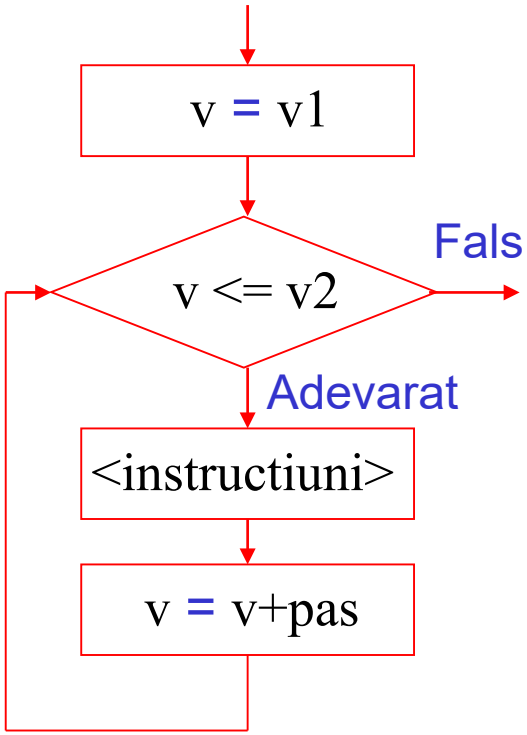
- uneori numărul de repetări ale corpului ciclului este cunoscut de la început
- în acest caz se poate folosi o variantă bazată pe o variabilă contor
- numărul de repetări: v2-v1+1 dacă pas=1

```
for  v = v1,v2,pas do
            <instructiuni>
endfor
```

⟺

```
v = v1
while v<=v2 do
            <instructiuni>
            v = v+pas
endwhile
```

# FOR - exemplu




$$\sum_{i=1}^{n} i = 1+2+\ldots+n = \sum_{i=1}^{n-1} i + n$$

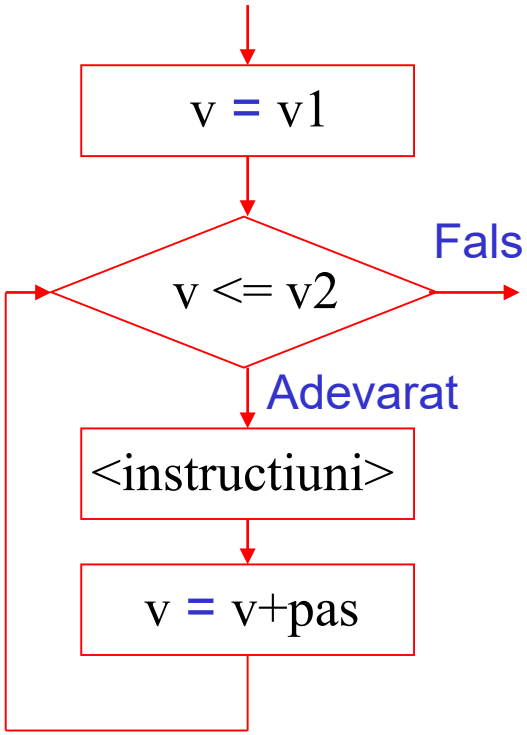
```
for  v = v1,v2,pas do
            <instructiuni>
endfor
```

```
S = 0   // pregătește variabila în
          //care se va colecta rezultatul

for i = 1,n do
   S = S+i  // adaugă termenul curent la S
Endfor

// noua valoarea = vechea valoare la care
     se adaugă termenul curent al sumei
```

# FOR - Python




Urmatoarea instructiune

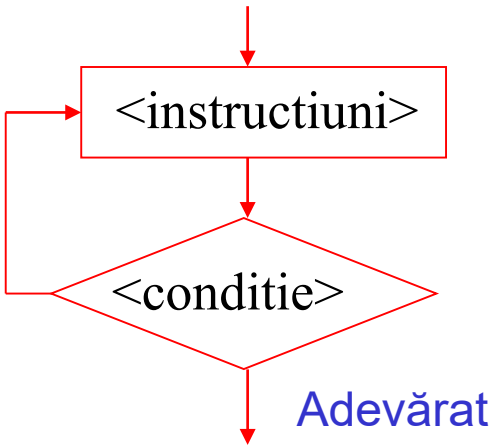
```
for  v = v1,v2,pas do
            <instructiuni>
endfor
```

```
for v in range(v1,v2+1,pas):
  <instructiuni>
```

Exemplu:

```
n=input("n=")
s=0
for i in range(1,n+1):
    s=s+i
print "s=",s
```

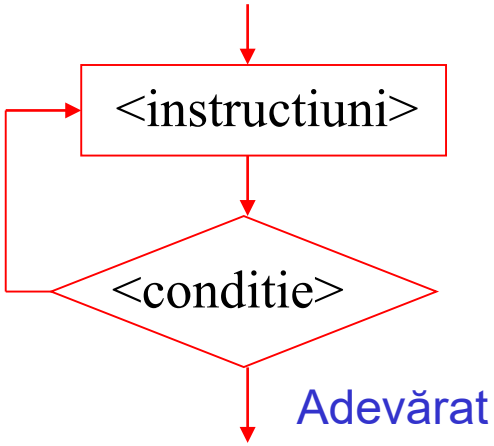
# REPEAT




```
repeat
    <instructiuni>
until <conditie>
```

- La inceput se execută corpul ciclului. Prin urmare acesta va fi executat cel puțin o dată
- Este analizată condiția de oprire iar dacă aceasta este falsă se execută din nou corpul ciclului
- Când condiția de oprire devine adevarată se trece la următoarea prelucrare a algoritmului
- Daca condiția de oprire nu devine niciodata adevarată atunci ciclul este infinit

# REPEAT - exemplu




```
repeat
    <instructiuni>
until <conditie>
```

$$\sum_{i=1}^{n} i = 1 + 2 + \ldots + n$$

```
S = 0
i = 1
repeat
   S = S+i
   i = i+1
until i>n
```

```
S = 0
i = 0
repeat
   i = i+1
   S = S+i
until i>=n
```

# REPEAT - exemplu

Observatie: orice instrucțiune de tip REPEAT poate fi rescrisă folosind WHILE prin:

- execuția explicită a corpului ciclului
- specificarea condiției de continuare prin negarea condiției de oprire de la REPEAT

```
repeat <instructiuni>
until <conditie>
```

```
<instructiuni>
while  NOT <conditie> do
    <instructiuni>
endwhile
```

$$\sum_{i=1}^{n} i = 1 + 2 + \ldots + n$$

```
S = 0
i = 0
repeat
   i = i+1
   S = S+i
until i>=n
```

```
S = 0
i = 0
while i<n
   i = i+1
   S = S+i
endwhile
```

# Sumar

- Algoritmii sunt proceduri de rezolvare pas cu pas a problemelor

- Trebuie să aibă proprietățile:
  - Corectitudine și generalitate
  - Finitudine
  - Rigurozitate (neambiguitate)
  - Eficiență

•Datele prelucrate de către un algoritm pot fi:
  - simple
  - structurate (ex: tablouri)

•Algoritmii pot fi descriși in pseudocod sau direct într-un limbaj de programare

# Sumar

- Pseudocod:

| | |
|---|---|
| Atribuire | ← sau := sau = |
| Transfer | read, write |
| Decizie | IF … THEN … ELSE … ENDIF |
| Ciclare | WHILE … DO … ENDWHILE |
| | FOR … DO … ENDFOR |
| | REPEAT … UNTIL |

- Python:

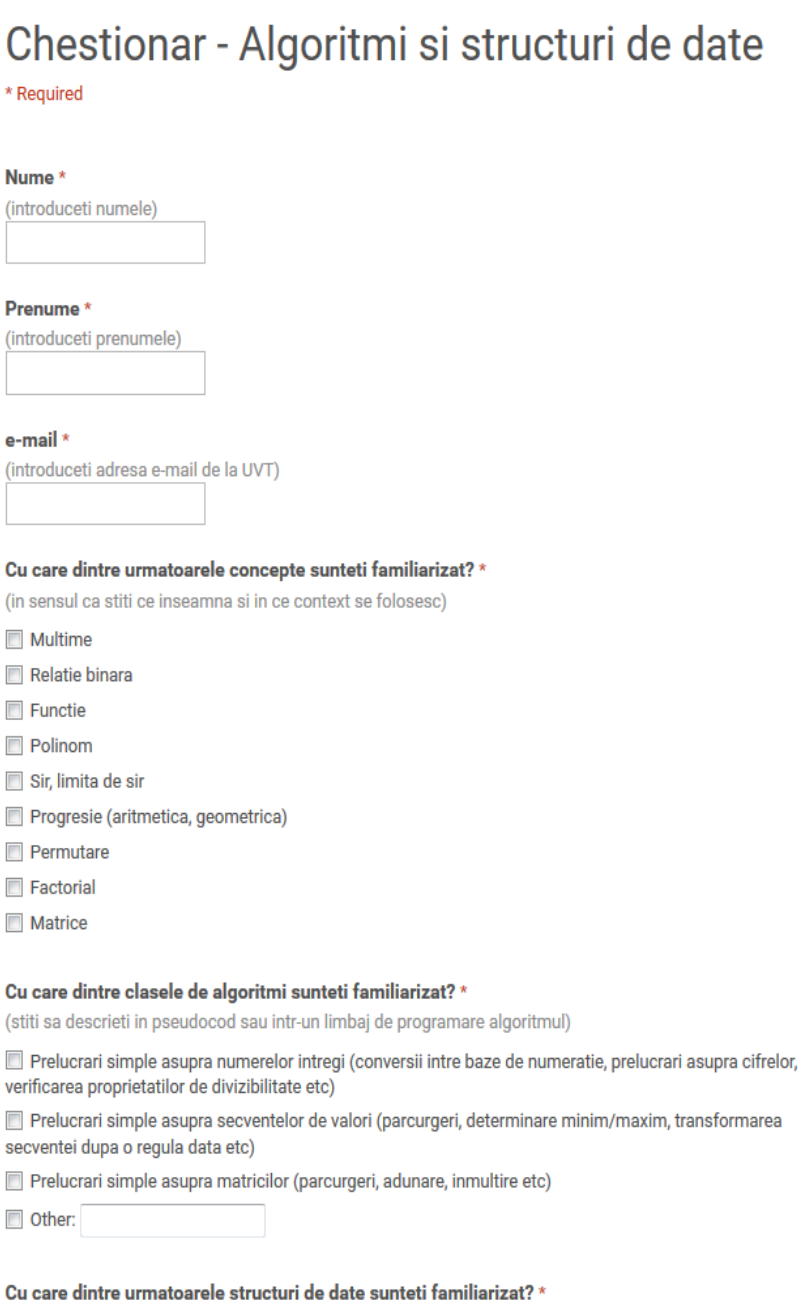
| | |
|---|---|
| Atribuire | = |
| Transfer | input, print |
| Decizie | if … elif … else |
| Ciclare | while:<br>for … in range .. |

# Următorul curs va fi despre …

- Subalgoritmi / functii
- Diferite exemple

# Chestionar

https://forms.gle/kjH3EXj6WrPv5HEu8



Chestionar - Algoritmi si structuri de date

* Required

**Nume** *

(introduceti numele)

**Prenume** *

(introduceti prenumele)

**e-mail** *

(introduceti adresa e-mail de la UVT)

**Cu care dintre urmatoarele concepte sunteti familiarizat?** *

(in sensul ca stiti ce inseamna si in ce context se folosesc)

- Multime
- Relatie binara
- Functie
- Polinom
- Sir, limita de sir
- Progresie (aritmetica, geometrica)
- Permutare
- Factorial
- Matrice

**Cu care dintre clasele de algoritmi sunteti familiarizat?** *

(stiti sa descrieti in pseudocod sau intr-un limbaj de programare algoritmul)

- Prelucrari simple asupra numerelor intregi (conversii intre baze de numeratie, prelucrari asupra cifrelor, verificarea proprietatilor de divizibilitate etc)
- Prelucrari simple asupra secventelor de valori (parcurgeri, determinare minim/maxim, transformarea secventei dupa o regula data etc)
- Prelucrari simple asupra matricilor (parcurgeri, adunare, inmultire etc)
- Other:

**Cu care dintre urmatoarele structuri de date sunteti familiarizat?** *