

Curs 1:

Introducere în rezolvarea algoritmică a problemelor

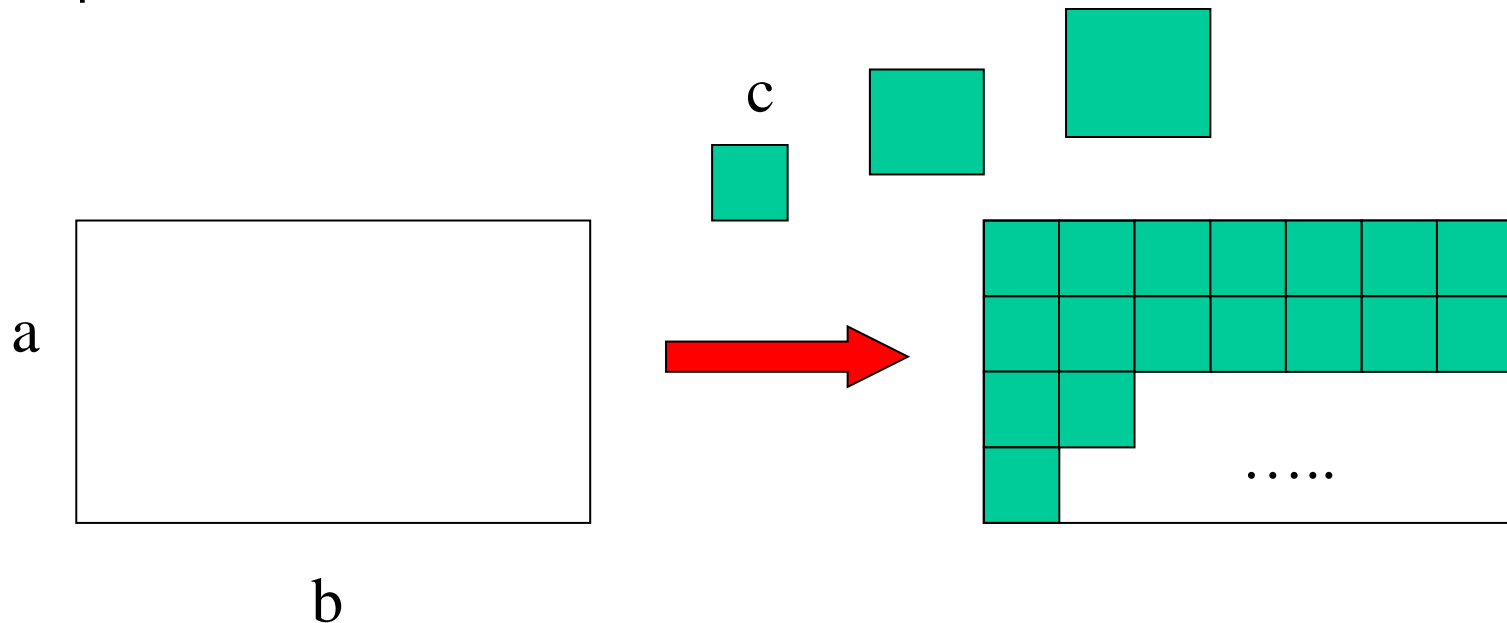
Cuprins

- Rezolvarea problemelor
- Ce este un algoritm ?
- Ce proprietăți ar trebui să aibă un algoritm ?
- Cum pot fi descriși algoritmi ?
- 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**

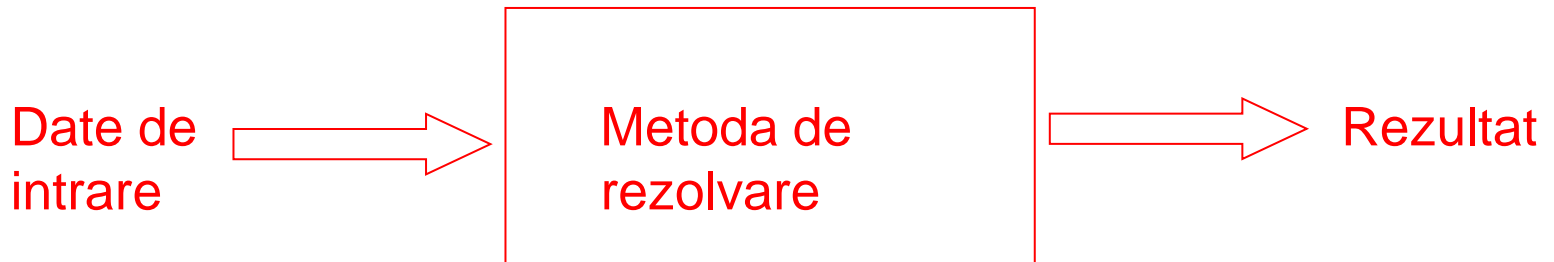
Rezolvarea problemelor

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

În 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 (în acest caz se asociază unei perechi de numere naturale valoarea celui mai mare divizor comun).
- O altă categorie de probleme e reprezentată de problemele care cer să se verifice dacă datele de intrare satisfac o anumită proprietate. Acestea sunt denumite **probleme de decizie**.

Exemplu: se verifică dacă un număr natural este prim sau nu

În 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 **algorithm**

Cuprins

- Rezolvarea problemelor
- Ce este un algoritm ?
- Ce proprietăți ar trebui să aibă un algoritm ?
- Cum pot fi descriși algoritmi ?
- 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)

↓
algorism → algorithm

- 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 împărțirii la un binom de forma $(X-a)$



Euclid
(cca. 325 -265 î.C.)

De la problemă la algoritm

Problema:

- Datele problemei
 - a, b - nr.naturale nenule
- Cerința:
 - determină $\text{cmmdc}(a,b)$

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 nenule
- **Cerinta:**
 - determina $\text{cmmdc}(a,b)$
- **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

Algoritm:

- **Variable** = obiecte abstracte ce corespund datelor problemei + datelor de lucru
 - deîmpărțit, împărțitor, rest
- **Secvența de prelucrări**
 1. Atribue deîmpărțitului valoarea lui a și împărțitorului valoarea lui b
 2. Calculează restul împărțirii deîmpărțitului la împărțitor
 3. Atribue deîmpărțitului valoarea împărțitorului și împărțitorului valoarea restului calculat la Pasul 2
 4. Dacă restul e nenul reia de la Pasul 2

De la algoritm la program

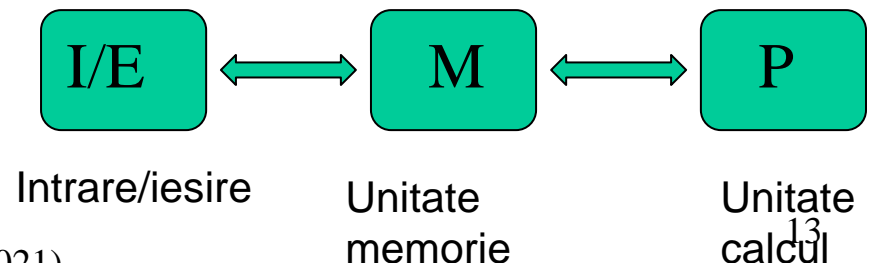
Algoritm:

- **Variable** = obiecte abstracte ce corespund datelor problemei + datelor de lucru
 - deîmpărțit, împărțitor, rest
- **Secvența de prelucrări**
 1. Atribuire deîmpărțitului valoarea lui a și împărțitorului valoarea lui b
 2. Calculează restul împărțirii deîmpărțitului la împărțitor
 3. Atribuire deîmpărțitului valoarea împărțitorului și împărțitorului valoarea restului calculat la Pasul 2
 4. Dacă restul e nenul reia de la Pasul 2

Program:

- **Variable** = 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



Cuprins

- Rezolvarea problemelor
- Ce este un algoritm ?
- Ce proprietăți ar trebui să aibă un algoritm ?
- Cum pot fi descriși algoritmi ?
- 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:

Pas 1: 2 ↔ 1 4 3 5

Pas 2: 1 2 4 3 5

Pas 3: 1 2 4 ↔ 3 5

Pas 4: 1 2 3 4 5

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 etape = 4 comparații)

Generalitate

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

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

În 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

Parcursere 1:

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

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

Parcursere 2:

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

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) = conduce, după un număr finit de operații la rezultatul corect pentru orice valori ale datelor de intrare care aparțin domeniului problemei
- Eficient

Finitudine

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

Exemplu

Pas 1: Asignează valoarea 1 lui a și valoarea 2 lui b ($a \leftarrow 1, b \leftarrow 2$)

Pas 2: modifică a: $a \leftarrow 3-a$

Pas 3: modifică b: $b \leftarrow 3-b$

Pas 4: Dacă $a+b > 1$ atunci se reia de la Pas 2 altfel STOP

Ce valoare are $a+b$ înainte de execuția pasului 4? Succesiunea de prelucrări se termină?

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 (spațiu de stocare, timp de execuție)

Eficiență

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

Exemplu: Se consideră un număr constituit din 10 cifre distincte.
Să se determine elementul următor din secvența crescătoare a numerelor naturale constituite din 10 cifre distincte.

0123456789, 0123456798,, 9876543210

0, 1, ..., 9999999999

Exemplu: $x = 6309487521 \rightarrow 6309512478$

Care este următorul număr (în ordine crescătoare) ce conține 10 cifre distincte?

Eficiență

Prima abordare:

Pas 1: se generează toate numerele în ordine crescătoare de la 0 la 9876543210

Pas 2: se caută în listă primul număr care are cifre distincte și este mai mare decât x

A doua abordare:

Pas 1: se generează doar numerele constituite din 10 cifre distincte

Pas 2: se ordonează crescător lista generată

Pas 3: se caută în listă primul număr care este mai mare decât x

Care variantă este mai eficientă?

Eficiență

Prima abordare:

Pas 1: se generează toate numerele în ordine crescătoare de la 0 la 9876543210

Câte numere sunt? 10 la puterea 10

Pas 2: se caută în listă primul număr care are cifre distincte și este mai mare decât x

Câte comparații se efectuează? cca 10 la putere 10

A doua abordare:

Pas 1: se generează doar numerele constituite din 10 cifre distincte

Câte numere sunt? $10! = 3628800$

Pas 2: se ordonează crescător lista generată (prin sortare rapidă)

Câte operații se efectuează? $10! \log(10!) = 2 \cdot 10^7$

Pas 3: se caută în listă primul număr care este mai mare decât x
 $\log(10! \log(10!))$

Eficiență

O scurtă analiză (imprecisă):

- Presupunem că avem la dispoziție un calculator care efectuează 10000 operații/secundă
- Pentru 10^{10} operații sunt necesare cca 277 ore adică cca 12 zile
- Pentru $10! = 3628800$ sunt necesare cca 6 minute

Obs:

- Fugaku (Japan): 7630848 nuclee, nr operatii 442 petaflops = $148.6 \cdot 10^{15}$ /s.
(<https://www.top500.org/lists/>)
- Ineficiența algoritmului nu poate fi întotdeauna compensată prin sporirea resurselor de calcul

Cuprins

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

Cum pot fi descriși algoritmi ?

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

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

Descriere algoritmică
?

Cum pot fi descriși algoritmi?

Există cel puțin următoarele modalități:

- **Scheme logice (diagrame):**
 - Descreri 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 algoritmi?

Există cel puțin următoarele modalități:

- Scheme logice (diagrame)
- Pseudocod:
- Limbaj de programare
 - Limbaj artificial în care pot fi descriși algoritmi 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>`

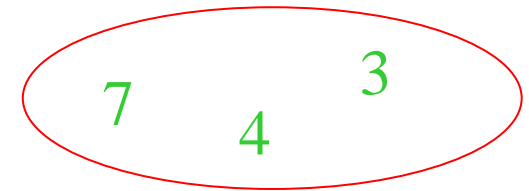
Obs. Există limbaje de programare (ex: Python) care nu necesită declararea explicită a datelor tipul variabilelor fiind stabilit în mod dinamic în momentul asignării unor valori

Ce tipuri de date pot fi utilizate ?

Date structurate: tablouri

Tablourile pot fi 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
- **Matrici**
 - Tablouri bidimensionale (multidimensionale)



3	7	4
---	---	---

Index: 1 2 3

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

(1,1)	(1,2)
1	0
0	1

(2,1) (2,2)

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

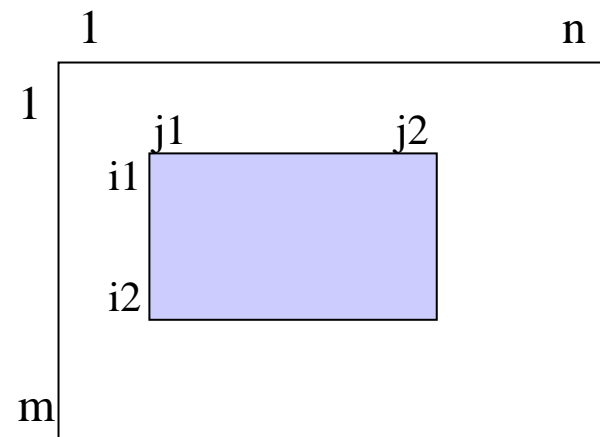
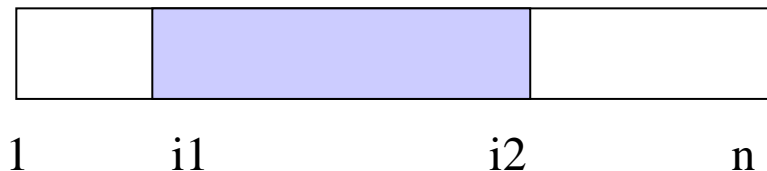
- Bidimensionale

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

Cum pot fi specificate datele ?

Specificarea subtablourilor

- **Subtablou**= porțiune contiguă a unui tablou
 - Unidimensional: $x[i1..i2]$ ($1 \leq i1 < i2 \leq n$)
 - Bidimensional: $A[i1..i2, j1..j2]$
($1 \leq i1 < i2 \leq m$, $1 \leq j1 < j2 \leq n$)



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)

(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 arbitrară

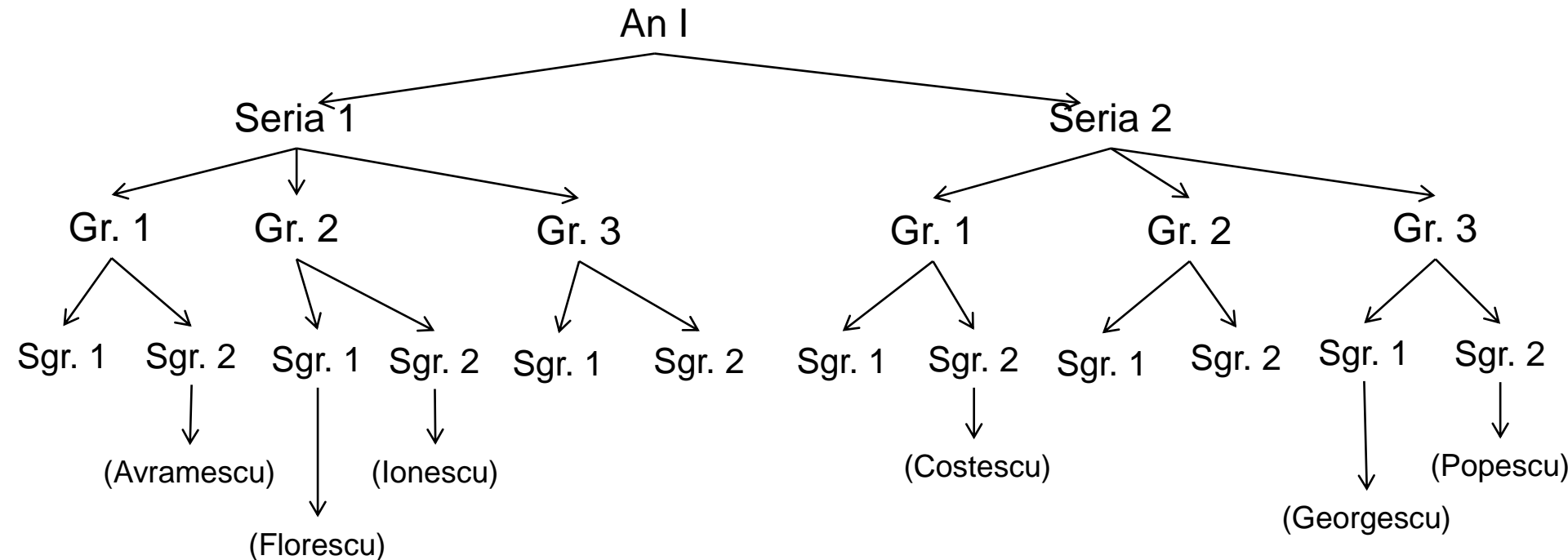
Ordine alfabetică

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

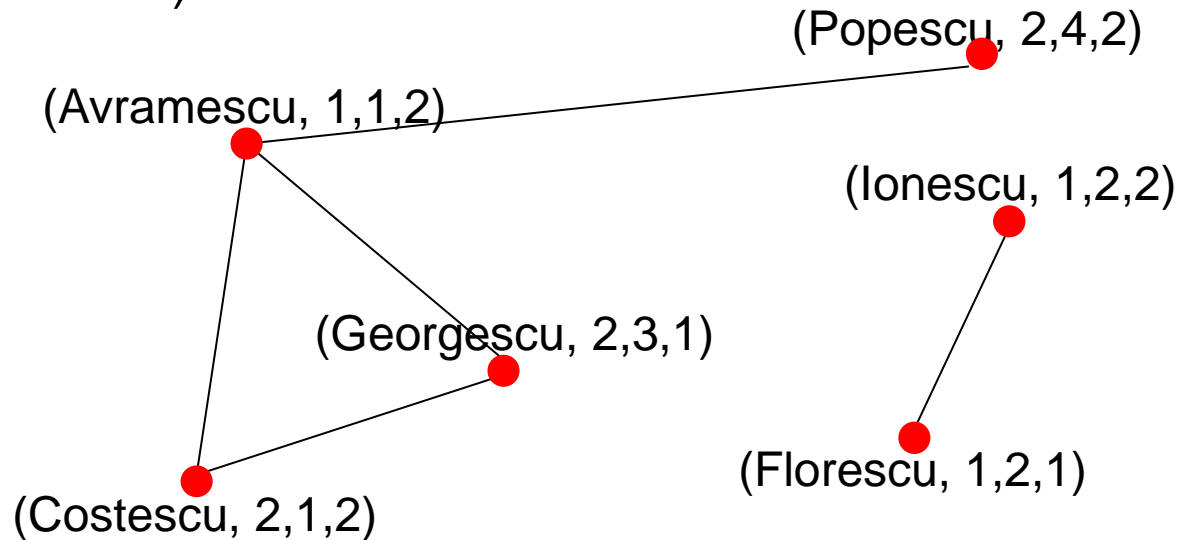


Date structurate

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 algoritmi ?
- 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:**
 $v \leftarrow \langle \text{expresie} \rangle$ sau $v := \langle \text{expresie} \rangle$ sau $v = \langle \text{expresie} \rangle$
- **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)

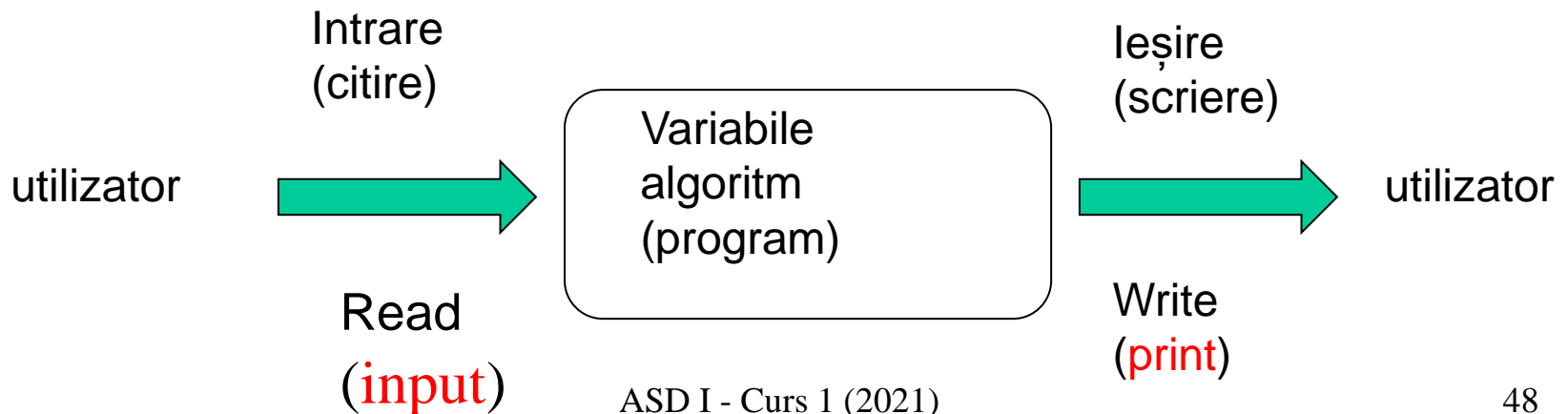
Obs:

- operatorii marcați cu roșu pot fi folosiți și în Python
- Cei marcați cu albastru pot fi folosiți doar în pseudocod

Intrare/ieșire

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

input v1,v2,...
print e1,e2,...



Intrare/ieșire

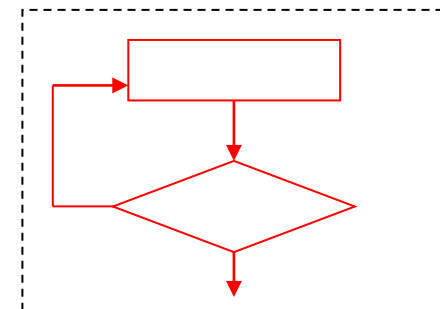
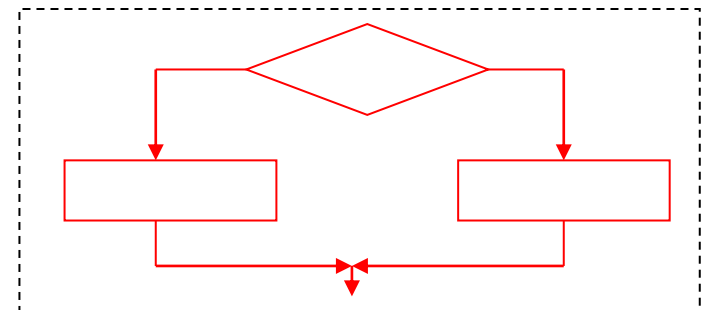
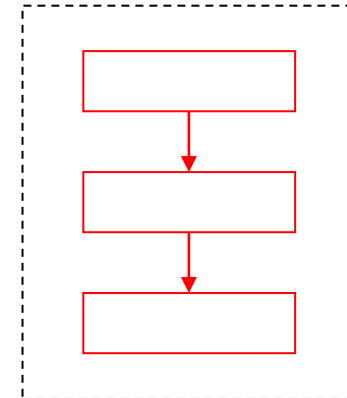
- Exemplu (Python)

```
# 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)

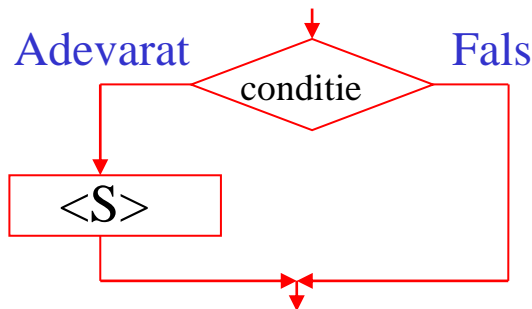
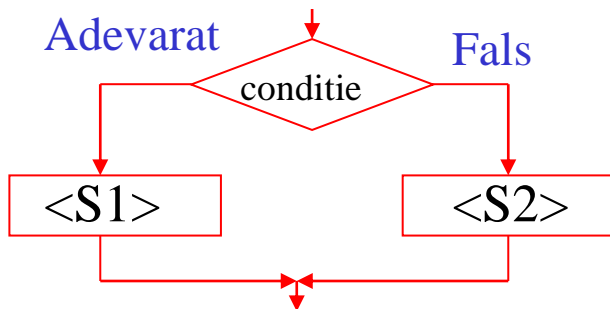
Structuri de prelucrare

- 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):**

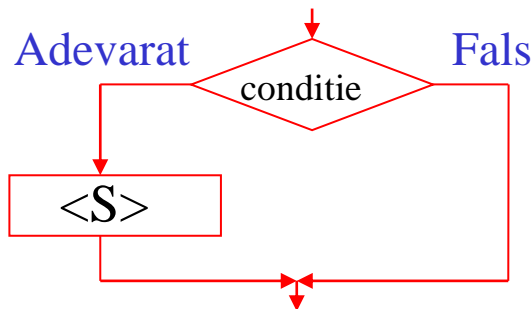
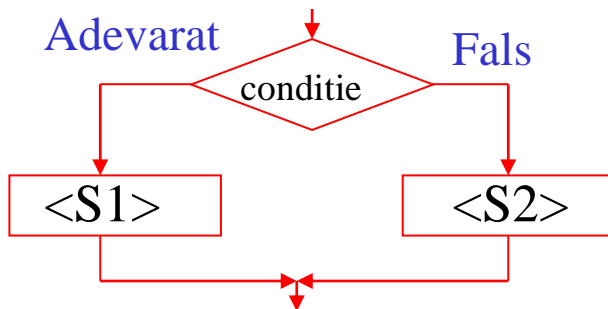
```
if <condiție> then <S1>
    else <S2>
endif
```

- **Varianta simplificată (pseudocod):**

```
if <condiție> then <S>
endif
```

Instrucțiune 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

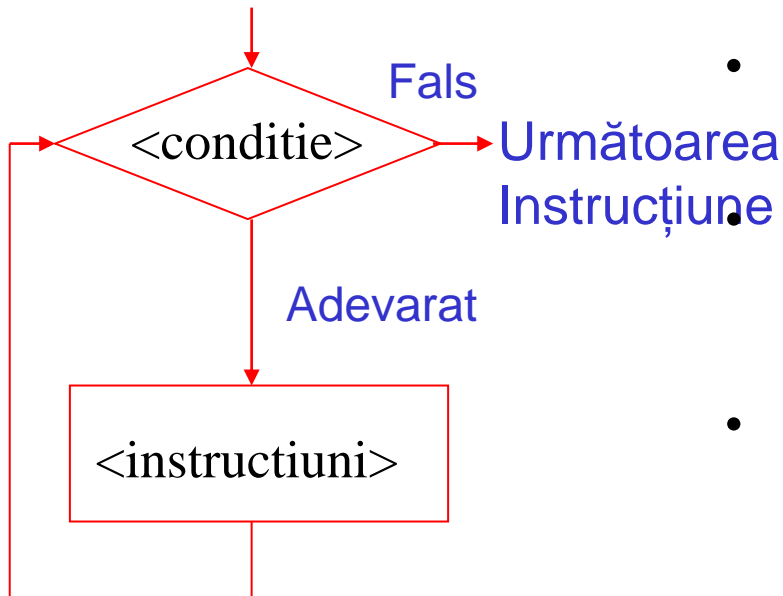
$$S(x) = \begin{cases} -1, & x < 0 \\ 0 & x = 0 \\ 1, & x > 0 \end{cases}$$

```
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
$$S = 1 + 2 + \dots + i + \dots + n$$
- 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)

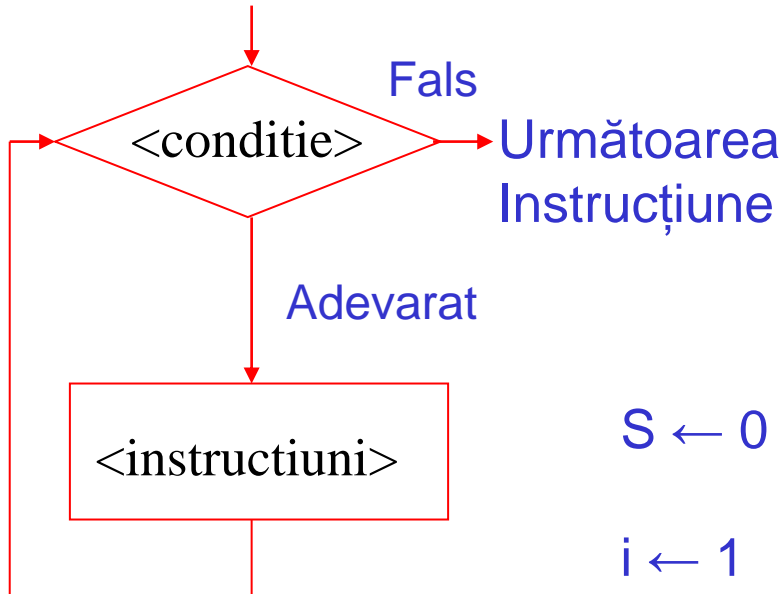
WHILE



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

- Se analizează condiția de continuare
- Dacă este adevărată 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



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

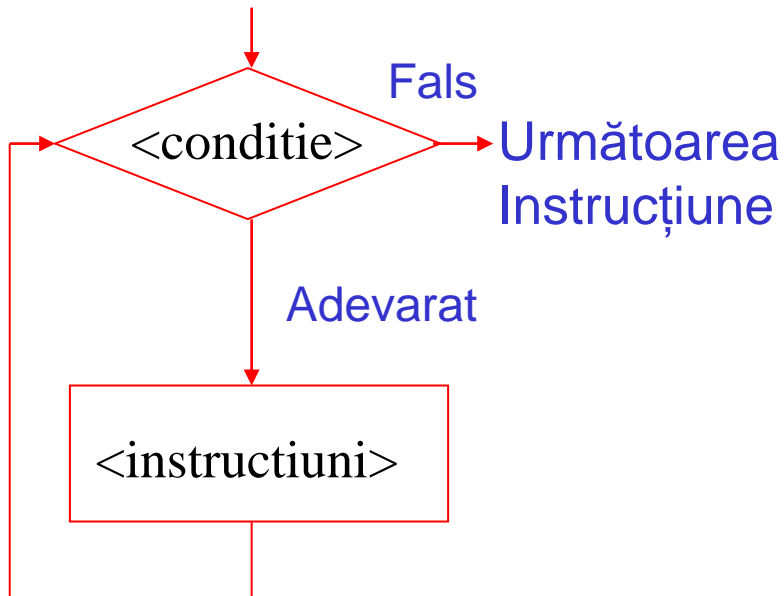
$S \leftarrow 0$ // pregătește variabila în
//care se va colecta rezultatul
 $i \leftarrow 1$ // inițializează indicele
// termenului de adăugat (termenul
// coincide cu indicele)

while $i \leq n$ do
 $S \leftarrow S + i$ // adaugă termenul la S
 $i \leftarrow i + 1$ // pregătește următorul
 // termen

endwhile

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


WHILE - Python



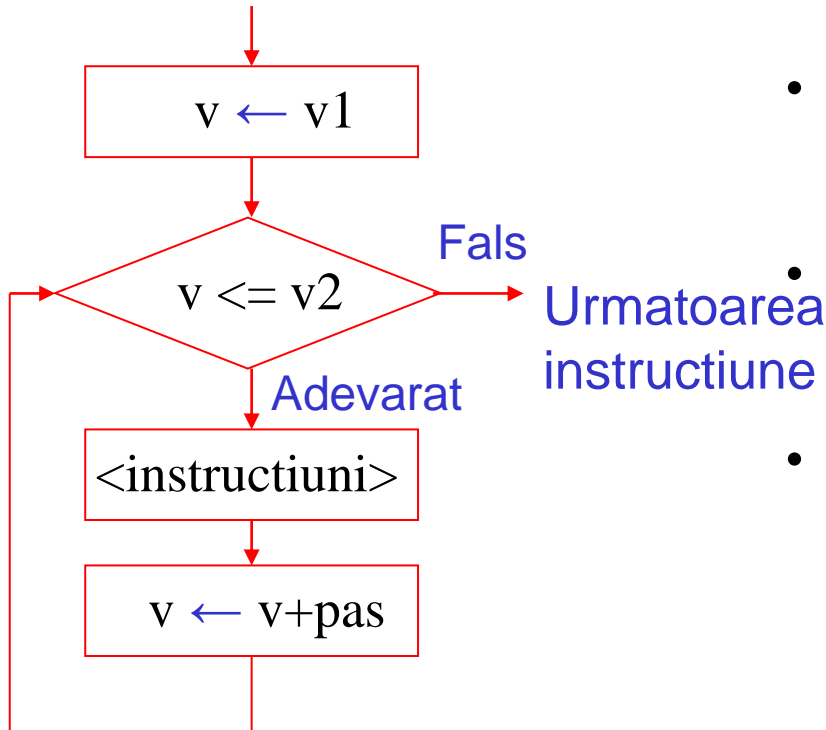
```
while <conditie> do
    <instructiuni>
endwhile
```

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

Exemplu:

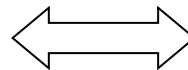
```
# Calculul unei sume
n=input("n=")
s=0
i=1
while (i<=n):
    s=s+i
    i=i+1
print "s=",s
```

FOR



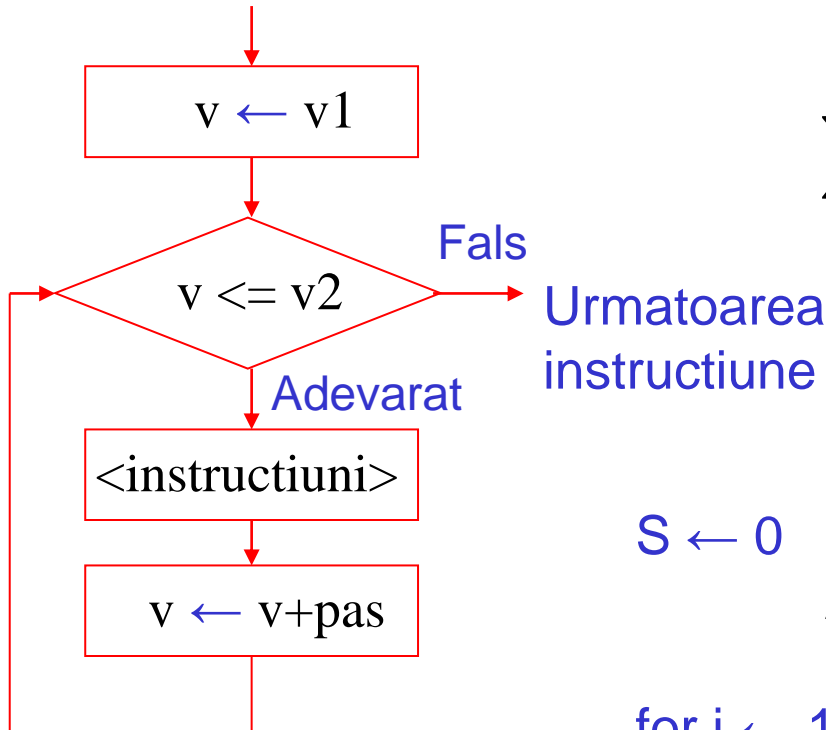
- uneori numărul de repetări ale corpului ciclului este cunoscut de la început
- în acest caz se poate folosi o variantă bazată pe o variabilă contor
- numărul de repetări: $v2 - v1 + 1$ dacă $\text{pas} = 1$

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



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

FOR - exemplu



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

$S \leftarrow 0$ // pregătește variabila în
//care se va colecta rezultatul

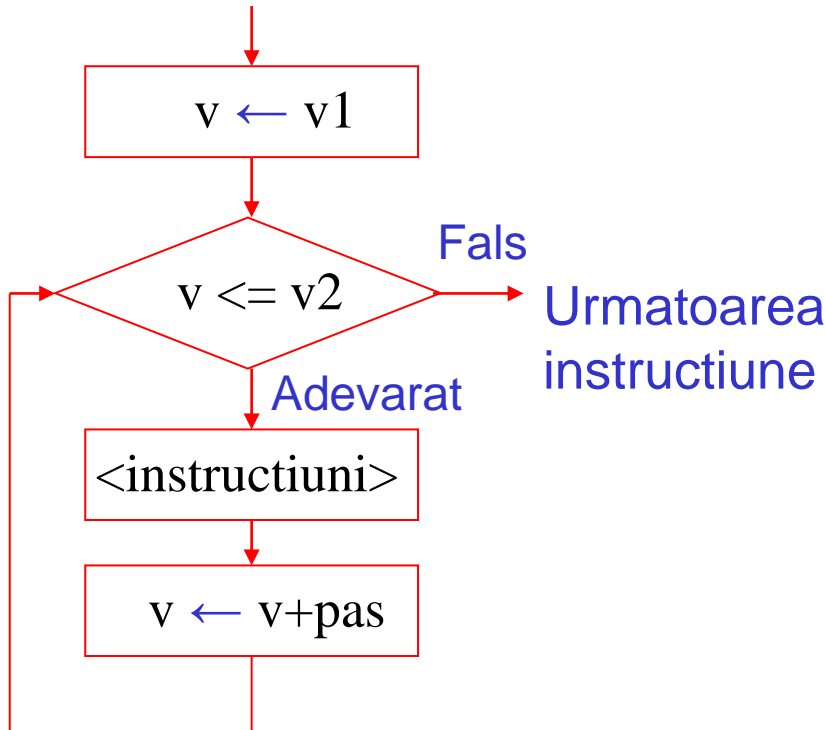
for $i \leftarrow 1, n$ do

$S \leftarrow S + i$ // adaugă termenul curent la S
Endfor

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

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

FOR - Python



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

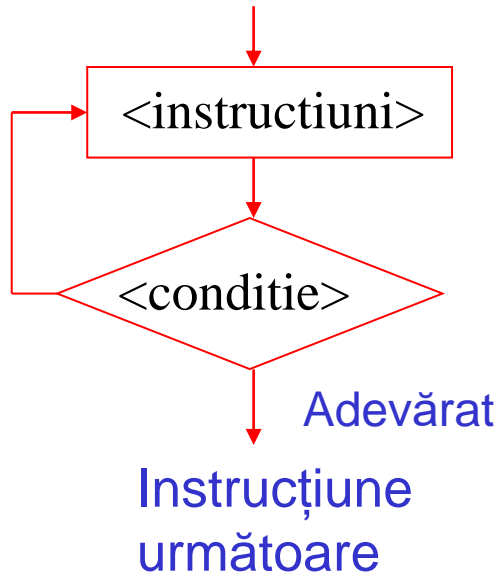
Exemplu:

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

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

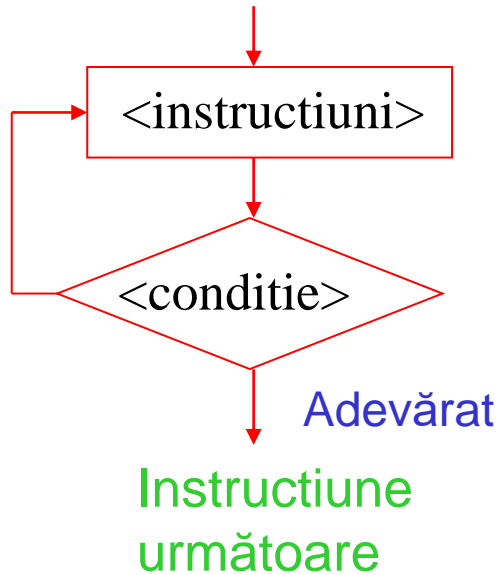
REPEAT

- La început 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 adevărată se trece la următoarea prelucrare a algoritmului
- Dacă condiția de oprire nu devine niciodată adevărată atunci ciclul este infinit



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

REPEAT - exemplu



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

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

```
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 <instrucțiuni>  
until <condiție>
```



```
<instrucțiuni>  
while NOT <condiție> do  
    <instrucțiuni>  
endwhile
```

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

```
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
 - Eficiență
 - Rigurozitate (neambiguitate)
- Datele prelucrate de către un algoritm pot fi:
 - simple
 - structurate (ex: tablouri)
- Algoritmii pot fi descriși în 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:
 for ... in range ..

Următorul curs va fi despre ...

- Subalgoritmi / functii
- Diferite exemple