Curs 1:

Introducere în rezolvarea algoritmică a problemelor

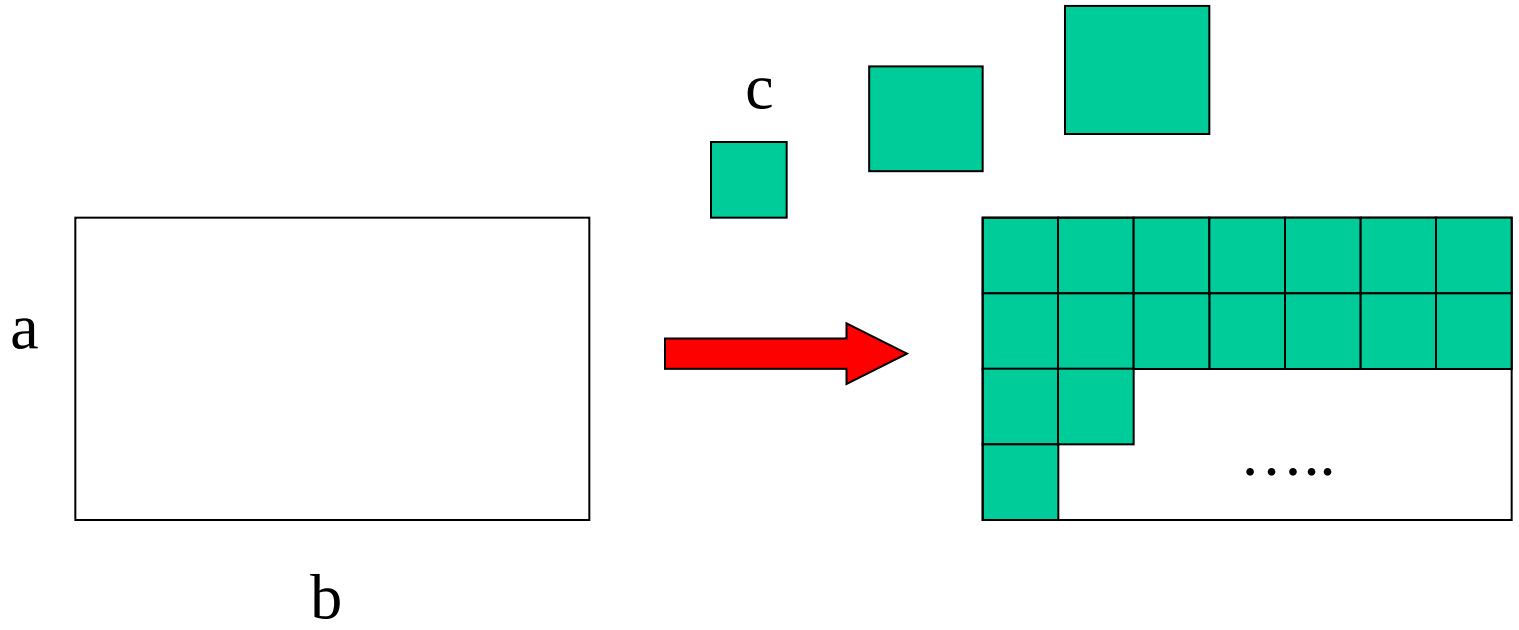
Cuprins

- 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

Universul 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**

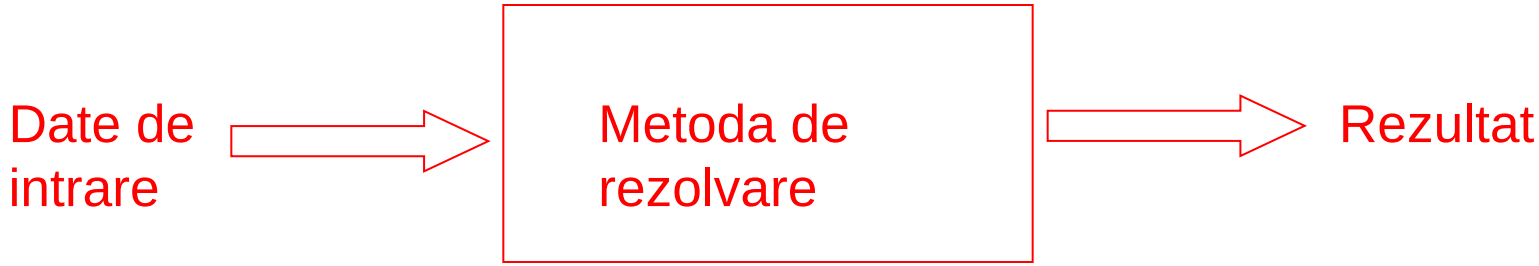
Rezolvarea problemelor

Problema = set de întrebări referitoare la anumite entități care reprezintă universul 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) si “ce se cere” (output)

Metoda de rezolvare = procedură 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 numar 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 prelucrari care pot fi executate de catre calculator … o astfel de metodă este un algoritm

Cuprins

- Rezolvarea problemelor
- Ce este un algoritm ?
- Ce proprietati ar trebui sa aiba un algoritm ?
- Cum pot fi descrisi algoritmii ?
- Ce tipuri de date vor fi utilizate ?
- Cum pot fi specificate prelucrarile 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 cuvantului ?

al-Khowarizmi - matematician persan (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 telefon, 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
- Algoritmul lui Horner
    - Calculul valorii unui polinom

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

De la problemă la algoritm

Problema:

- Datele problemei
  - a, b - nr.naturale

- Metoda de rezolvare
  - Imparte a la b și retine restul
  - Imparte b la rest și reține noul rest
  - continuă impărțirile până se ajunge la un rest nul
  - Ultimul rest nenul reprezintă rezultatul

De la problema la algoritm

Problema:

- Datele problemei
  - a, b - nr.naturale
- Metoda de rezolvare
  - Imparte a la b și retine restul
  - Imparte 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
  - deimpărțit, împărțitor, rest
- Secvența de prelucrări
  1. Atribuie deimpărțitului valoarea lui a și împarțitorului 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

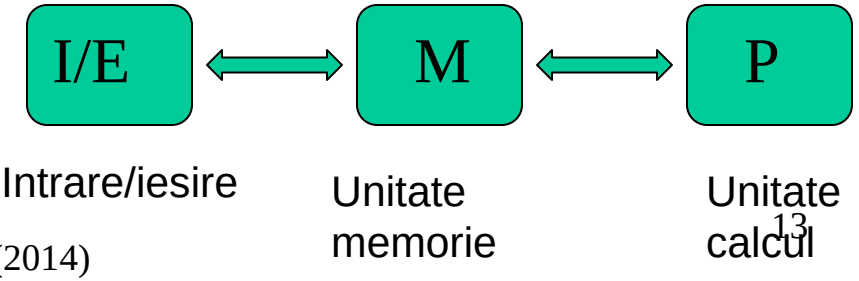
Algoritm:

- Variabile = obiecte abstracte ce corespund datelor problemei
  - deimpărțit, împărțitor, rest
- Secvența de prelucrări
  1. Atribuie deimpărțitului valoarea lui a și împartitorului 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



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 ?

- Generalitate
- Finitudine
- Neambiguitate
- Eficienta

Generalitate

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

Exemplu:

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

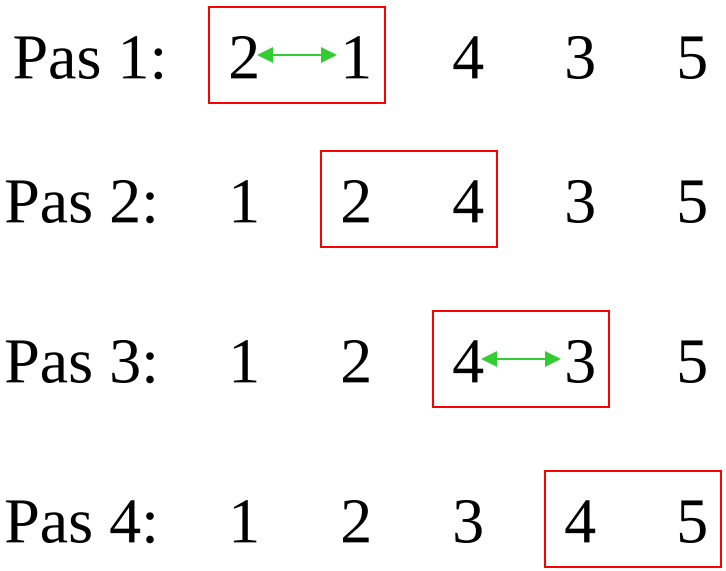
De exemplu:

(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 completa 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 ?

- Generalitate
- Finitudine
- Neambiguitate
- Eficiență

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;
- Pas2: Adaugă 2 la x;
- Pas3: Dacă x=10 atunci STOP;
- altfel afișează x; se reia de la Pasul 2

x=1

x=3 x=5 x=7 x=9 x=11

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 ?

- Generalitate
- Finitudine
- Neambiguitate
- Eficiență

Neambiguitate

Operațiile dintr-un algoritm trebuie definite în mod riguros:

- La execuția fiecărui pas trebuie specificat clar ce trebuie executat și care va fi următorul pas

Exemplu:

Pas 1: Atribuie 1 lui x

Pas 2: Fie incrementează x cu 1 fie decrementează x cu 1

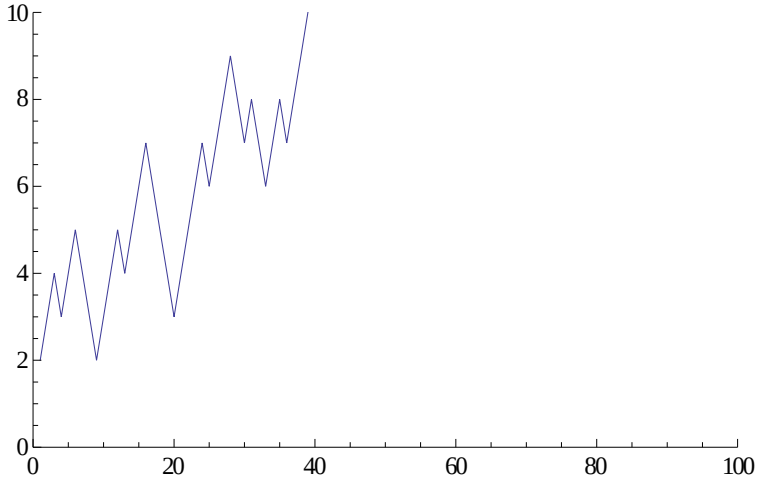
Pas 3: Dacă $x \in [1,10]$ atunci se reia de la Pasul 2; altfel Stop.

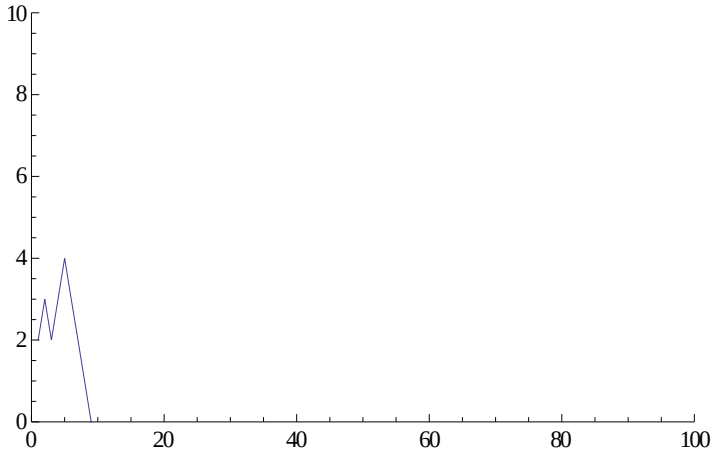
Atât timp cât nu este specificat un criteriu clar în baza căruia să se decidă dacă x este incrementat sau decrementat secvența de mai sus nu poate fi considerată algoritm

Neambiguitate

Exemplu:

Pas 1: Atribuie 1 lui x
Pas 2: Fie incrementează x cu 1 fie decrementează x cu 1
Pas 3: Dacă x∈[1,10] atunci se reia de la Pasul 2; altfel Stop.





Neambiguitate

Modificăm algoritmul anterior după cum urmează:

Pas 1: Atribuie 1 lui x

Pas 2: Aruncă o monedă

Pas 3: Daca se obține pajură

atunci incrementează x cu 1

altfel decrementează x cu 1

Pas 4: Dacă $x \in [1,10]$ atunci se reia de la Pasul 2, altfel Stop.

- De această dată algoritmul poate fi executat dar … la rulări diferite poate conduce la rezultate diferite
- Acesta este un exemplu de algoritm aleator
- Introducerea elementelor aleatoare in algoritmi poate facilita gasirea unor solutii

Ce proprietăți ar trebui să aibă un algoritm ?

- Generalitate
- Finitudine
- Neambiguitate
- Eficiență

Eficiență

Un algoritm trebuie să folosească un volum rezonabil de resurse de calcul: memorie și timp de calcul

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

Exemplu: O istorioară (plauzibilă sau nu ...) :

Să presupunem că am întâlnit pe cineva la o conferință, persoana respectivă mi-a dat o carte de vizită dar ... am pierdut-o. Imi amintesc doar faptul că numărul de telefon era din 10 cifre distincte. Să fi fost 5634890127 sau 8961073245 sau ... ?

Sunt insă convinsă că dacă aș vedea numele mi-aș aminti.

Am o carte de telefon electronică (care permite atât căutare după nume în ordine alfabetică cât și căutare în ordinea crescătoare a numerelor).

Cum pot proceda?

Eficiență

Prima abordare:

Pas 1: generează toate numerele de telefon care pot fi obținute folosind cele 10 cifre

Pas 2: pentru fiecare număr de telefon generat se caută în cartea de telefon

A doua abordare:

se parcurge cartea de telefon în ordinea alfabetică a numelor persoanelor și pentru fiecare persoană se verifică dacă numărul de telefon conține cifre distincte

Care variantă este mai bună (în raport cu numărul de operații efectuate) ?

Eficiență

Prima abordare:

Pas 1: generează toate numerele de telefon care pot fi obținute folosind cele 10 cifre

Pas 2: pentru fiecare număr de telefon generat se caută în cartea de telefon

Notăm cu:

- m = numărul de cifre ale numărului (ex: m=10)
- n = numărul de înregistrări în cartea de telefon

O estimare a numărului de operații elementare (comparații):

- Numărul de numere de telefon posibile: m!
- Numărul de comparații pentru fiecare număr de telefon: n sau $\log_2 n$
- Numarul de cifre comparate pentru fiecare număr de telefon: m

$m! * m * \log_2 n$

Eficiență

A doua abordare:

se parcurge cartea de telefon în ordinea alfabetică a numelor persoanelor și pentru fiecare persoană se verifică dacă numărul de telefon conține cifre distincte

Estimarea numărului de comparații:

- Parcurgerea secvențială a n înregistrări și verificarea dacă numărul conține doar cifre distincte - cel mult m^2 comparații

$n\ m^2$ (în variantă tip forță brută)

sau

nm (bazat pe tabel de frecvențe a cifrelor)

Eficiență

Care abordare este mai buna ?

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

Exemplu: m=10

n= 304.467 (populația Timișoarei în 2011)

Aproximativ

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

Eficiență

Care abordare este mai buna ?

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

Exemplu: m=10

n= 304.467 (populația Timișoarei în 2011)

Aproximativ

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

Dar daca m=20 ...

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

Eficiență

O scurta analiză (imprecisă):

$8.8*10^{20}$ operații înseamnă aproximativ $7*10^{7}$ secunde pe un supercalculator 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 2013: Tianhe-2 (China) – $33.86*10^{15}$ operații/secundă

BG/P @ UVT

http://hpc.uvt.ro

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

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

Descriere algoritmică

?

Cum pot fi descriși algoritmii ?

Există cel puțin următoarele modalități:

- Scheme logice:
  - 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)
    - sintaxa (set de reguli de construire a frazelor limbajului)
  - Nu e la fel de restrictiv ca un limbaj de programare
- Limbaj de programare
  - Limbaj artificial in care pot fi descrisi algoritmii pentru a putea fi executati de catre 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:

- 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:

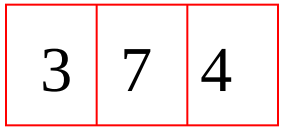
- nume
- valoare
  - constantă (aceeași valoare pe parcursul executiei algoritmului)
  - variabilă (valoarea se schimbă pe parcursul execuției algoritmului)
- tip
  - simplu (ex: numere, caractere, valori de adevăr …)
  - structurat (ex: tablouri)

Ce tipuri de date pot fi utilizate ?

Tablourile sunt utilizate pentru a reprezenta:

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

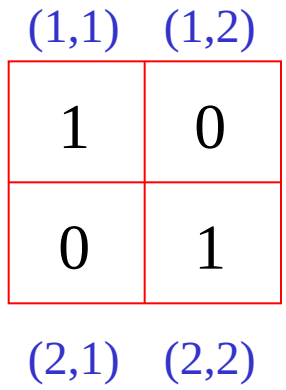
- Secvențe (obs:. (3,7,4) este diferit de (3,4,7))
  - Ordinea elementelor are importanță



Index: 1 2 3

- Matrici
  - Tablouri multidimensionale

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



Cum pot fi specificate datele ?

- Date simple:
  - Intregi INT <nume variabila>
  - Reale REAL <nume variabila>
  - Logice BOOLEAN <nume variabila>
  - Caractere CHAR <nume variabila>

Obs. Există limbaje de programare (ex: Python) care nu necesită declararea explicită a datelor

Cum pot fi specificate datele ?

Tablouri

Unidimensionale

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

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

Bi-dimensionale

<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

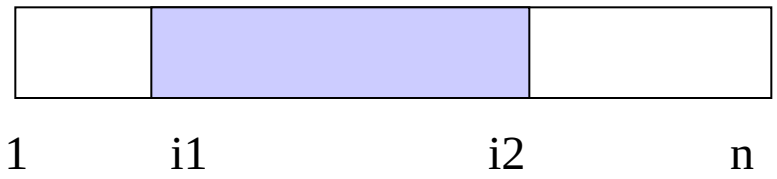
  A[i,j] - i este indice de linie, j este indice de coloană

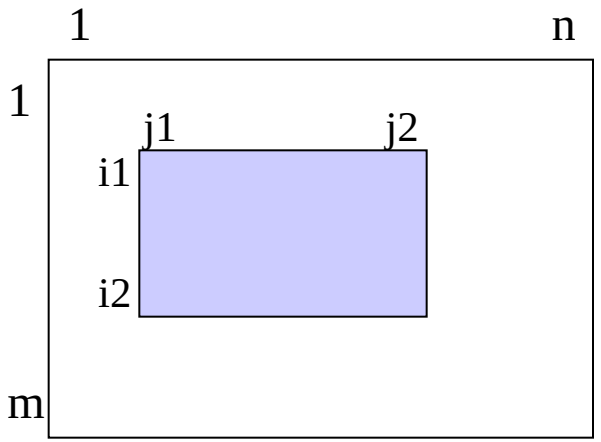
Cum pot fi specificate datele ?

Specificarea subtablourilor

- Subtablou= portiune 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)





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 prelucrarile dintr-un algoritm ?

Instrucțiune

= acțiune (operație) executată de către un algoritm

Tipuri de instrucțiuni:

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

Atribuire

- Scop: atribuie o valoare unei variabile
- Descriere:

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 (NOT, OR, AND)

Operatori

- Aritmetici:
  + (adunare), - (scădere), *(înmulțire),
  / (impartire), ^ (ridicare la putere),
  DIV sau / (câtul împartirii 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:

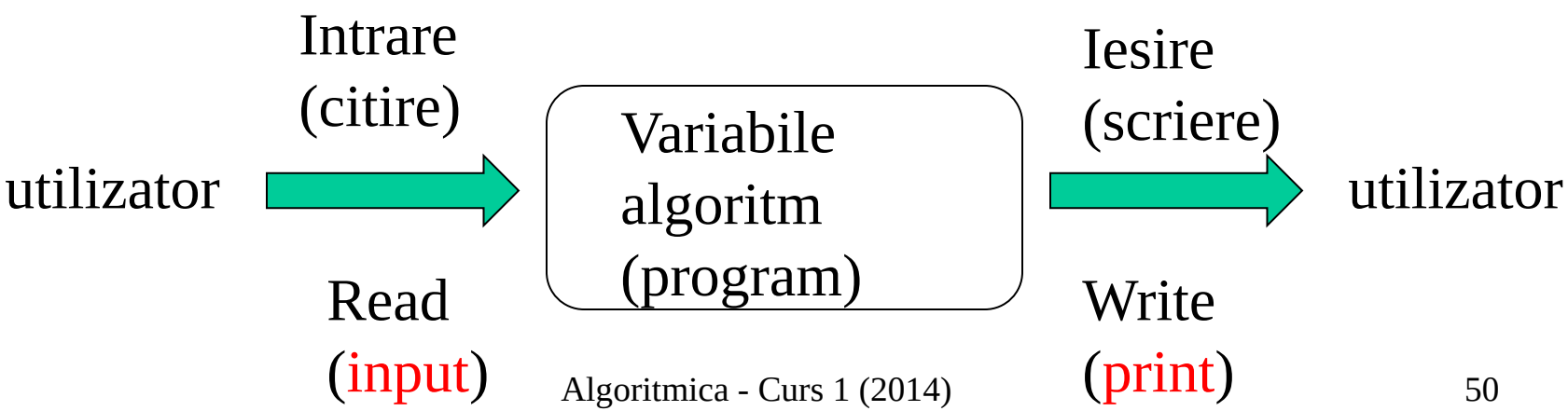
- operatorii marcati cu rosu pot fi folositi si in Python
- Cei marcati cu albastru pot fi folositi doar in pseudocod

Intrare/iesire

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

read v1,v2,…

write e1,e2,…

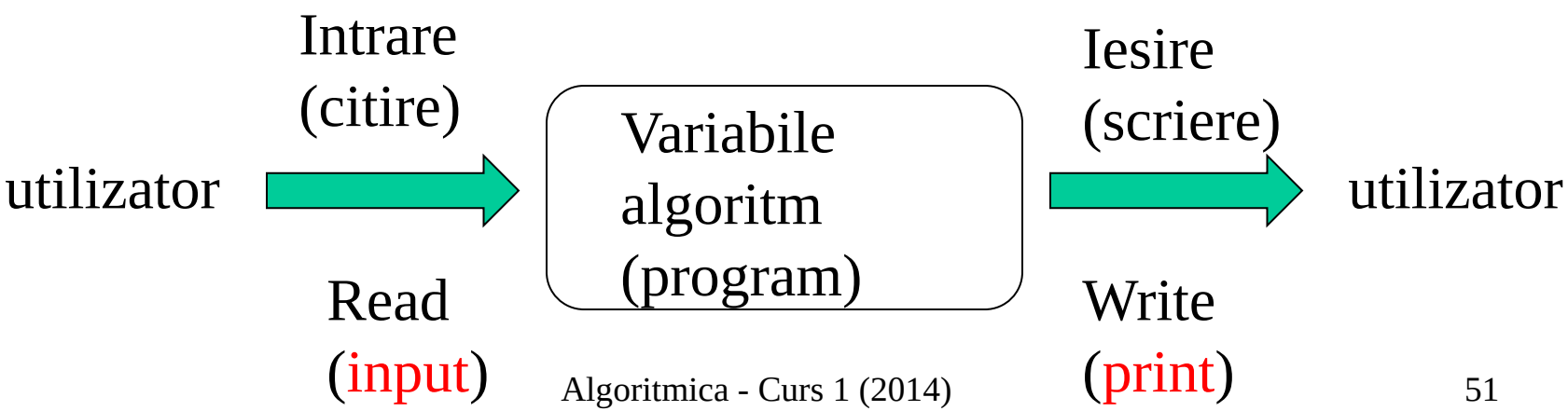


Intrare/iesire

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

read v1,v2,…

write e1,e2,…



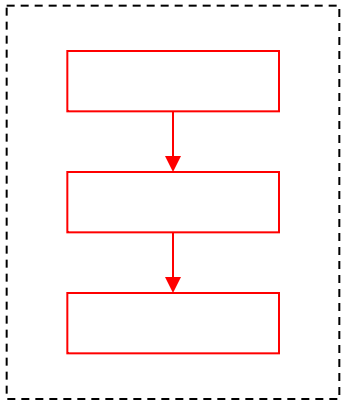
Intrare/iesire

- Exemplu (Python)

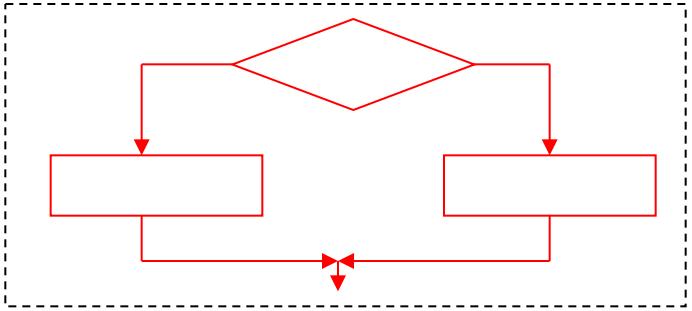
```
# Calculul ariei si perimetrului unui dreptunghi
a=input("Lungime dreptunghi=")  # preluare date de intrare
b=input("Largime dreptunghi=")
aria=a*b                        # calcul arie
perimetru=2*(a+b)               # calcul perimetru
print "Arie=", aria             # afisare rezultate
print "Perimetru=",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

Structuri de prelucrare

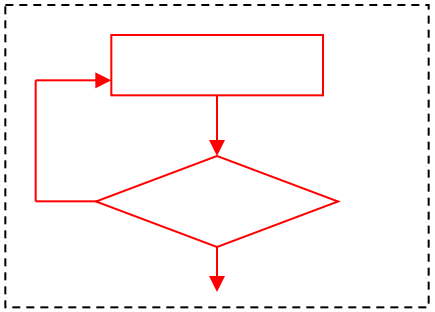
- Secvența de instrucțiuni



- Instrucțiune de decizie (condițională)

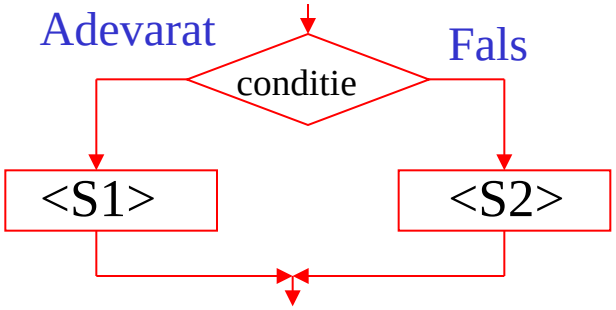


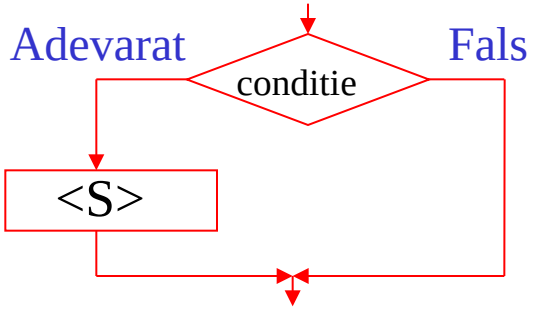
- Instrucțiune de ciclare (repetitivă)



Instructiune de decizie

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





- Varianta generală (pseudocod):

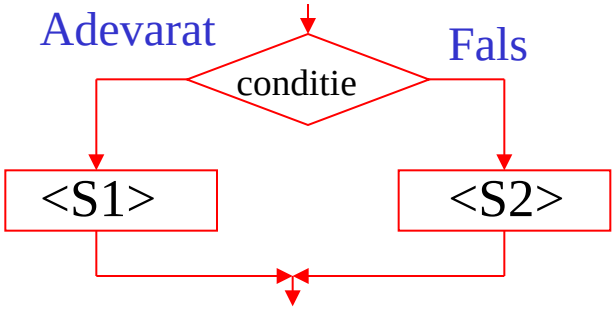
```
if <condiție> then  <S1>
                else  <S2>
endif
```

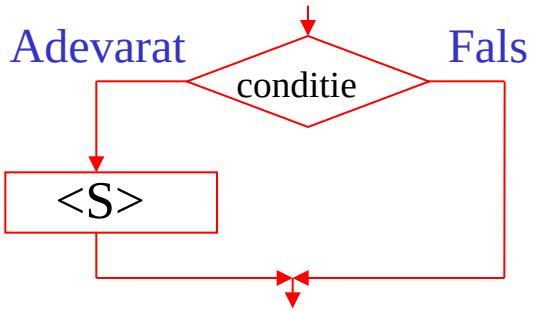
- Varianta simplificată (pseudocod):

```
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 daca un numar 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
```

Instructiuni de ciclare

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

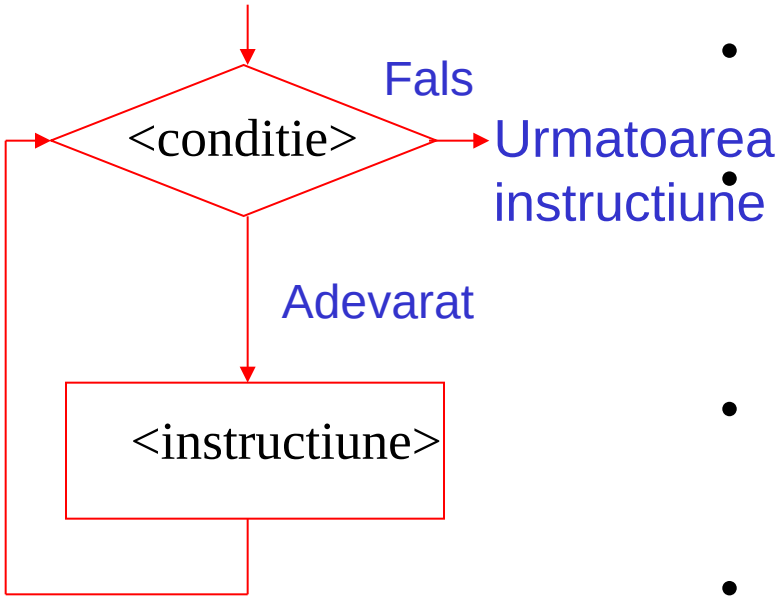
  S= 1+2+…+i+…+n
- Un ciclu este caracterizat prin:
  - Pasul de prelucrare care trebuie repetat

    (ex: adunarea următoarei valori la valoarea curenta 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 precondiționate (WHILE)
  - Cicluri postcondiționate (REPEAT)

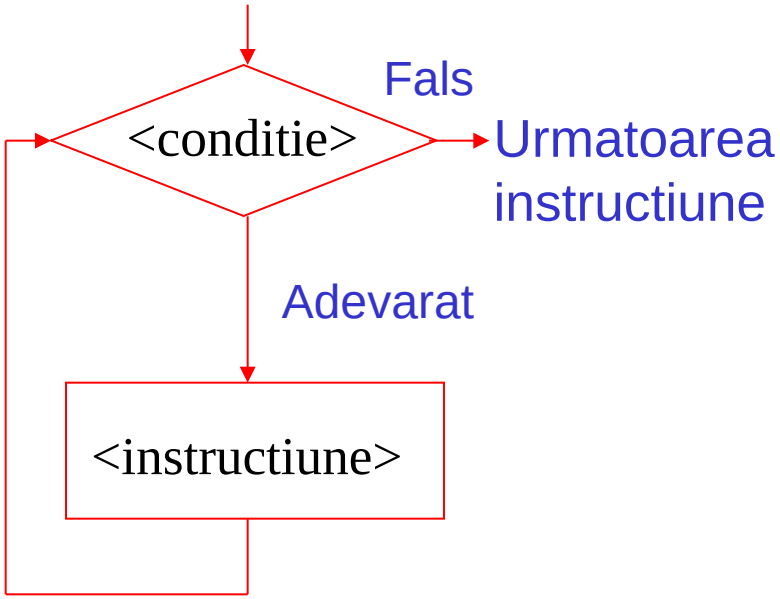
WHILE



- 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  <condiție> do
        <instrucțiune>
endwhile
```

WHILE - exemplu

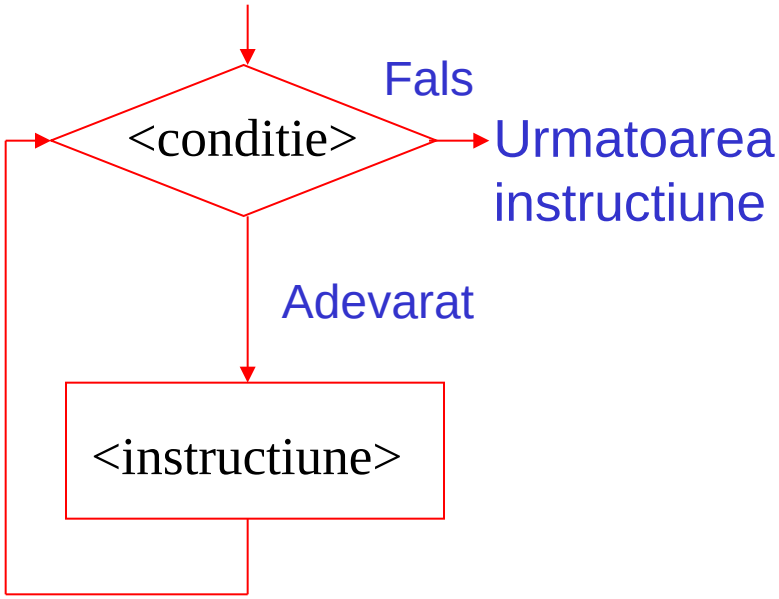


$$\sum_{i=1}^{n} i = 1 + 2 + ... + n$$

```
while  <conditie> do
        <instructiune>
endwhile
```

```
S=0   // pregateste variabila in
       //care se va colecta rezultatul
i=1   // initializeaza indicele
       // termenului de adaugat
while i<=n do
   S=S+i  // adauga termenul la S
   i=i+1   // pregateste urmatorul
           //termen
endwhile
```

WHILE - Python



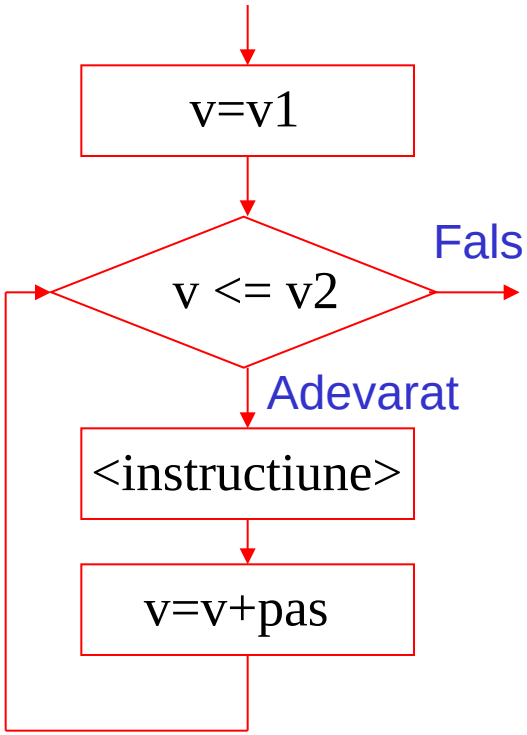

```
while <conditie>:
  <instructiuni>
```

while <conditie> do
<instructiune>
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

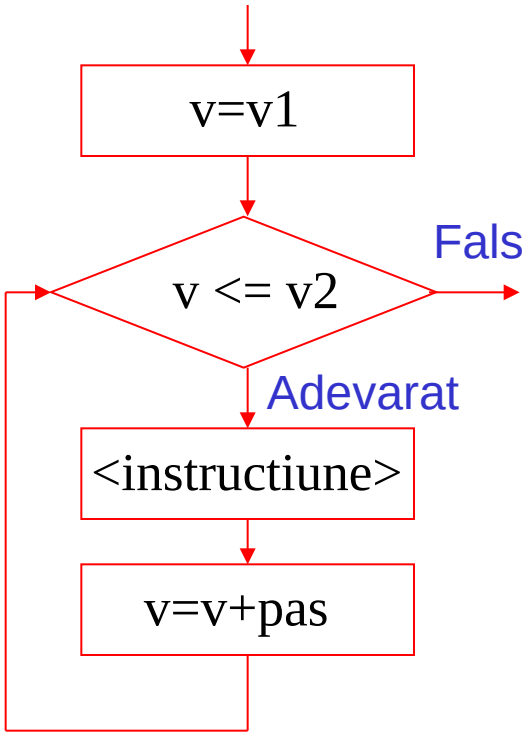
- Uneori numărul de repetări ale corpului ciclului este cunoscut de la început
- In acest caz se poate folosi o variantă bazată pe o variabilă contor
- Numărul de repetări: v2-v1+1 dacă pas=1

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

⟺

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

FOR - exemplu



Urmatoarea instructiune

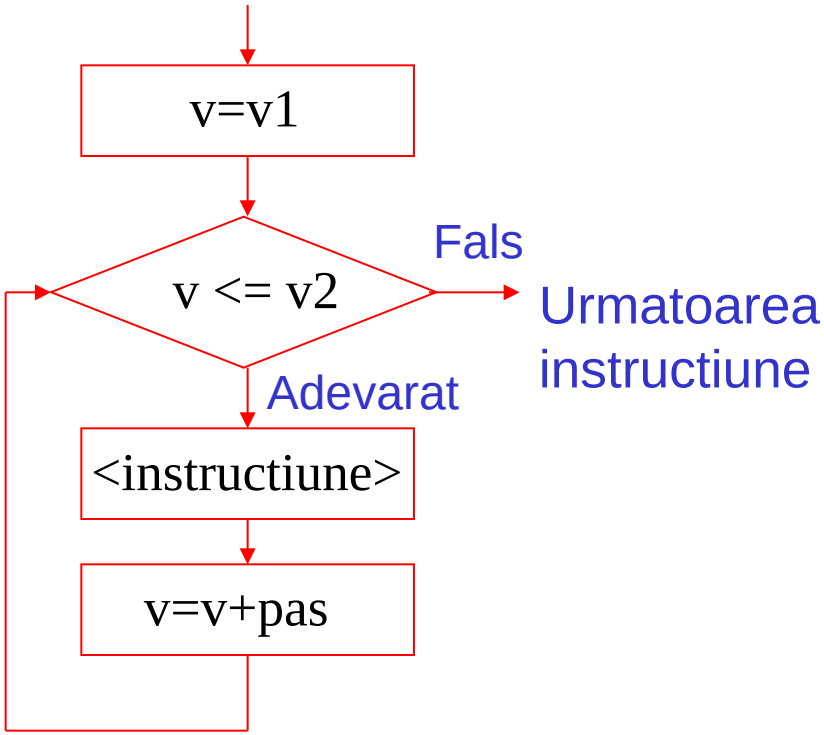
$$\sum_{i=1}^{n} i = 1 + 2 + ... + n$$

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

```
S:=0   // pregateste variabila in
       //care se va colecta rezultatul

for i=1,n do
   S=S+i  // adauga termenul curent la S
endfor
```

FOR - Python



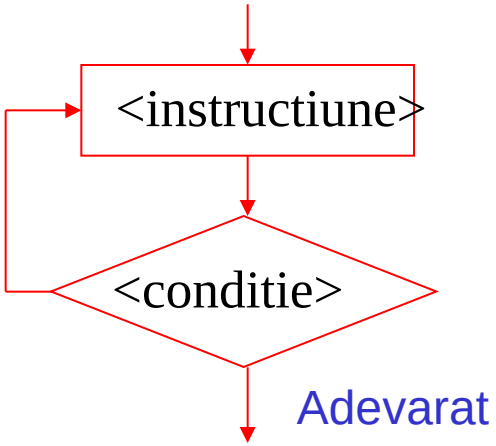

```
for v in range(v1,v2,pas):
  <instructiune>
```

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
          <instructiune>
endfor
```

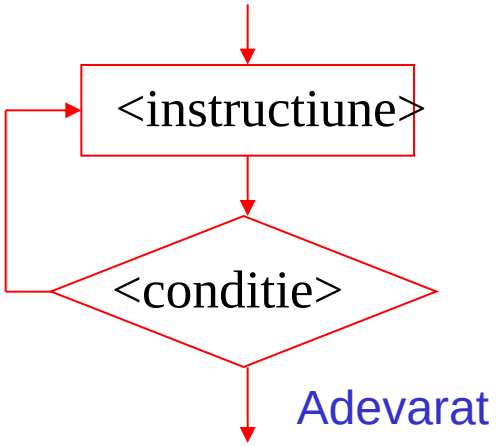
REPEAT




```
repeat
    <instructiune>
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 adevarata atunci ciclul este infinit

REPEAT - exemplu



repeat
    <instructiune>
until <conditie>

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

```
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 <instructiune>
until <conditie>
```

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

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

```
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 intr-un limbaj de programare

Sumar

- Pseudocod:

Atribuire	← 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

Chestionar

http://goo.gl/sUR1XQ

Chestionar Algoritmica

*Obligatoriu

Nume *
(introduceti numele)

Prenume *
(introduceti prenumele)

e-mail *
(introduceti adresa e-mail de la UVT)

Cu care dintre clasele de algoritmi sunteti familiarizat? *
(altii sa descrieti in pseudocod sau 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)

Cu care dintre algoritmii de sortare sunteti familiarizati? *

- sortare prin insertie
- sortare prin selectie
- sortare prin interschimbarea elementelor vecine (bubble sort)
- sortare prin interclasare
- sortare rapida (quicksort)
- Altele:

Cu care dintre urmatoarele tehnici de proiectare a algoritmilor sunteti familiarizati ? *

- Divizarea problemei (divide et impera / divide and conquer)
- Alegere lacoma (greedy)
- Programare dinamica
- Cautare cu revenire (backtracking)

Cu ce limbaj de programare sunteti familiarizati? *
(in care ati implementat/testat cel putin un program)

- C
- C++
- Pascal
- Java
- Python
- Altele:

Sunteti familiarizati cu analiza complexitatii algoritmilor? *

- Putin (pot aprecia daca un algoritm este mai costisitor decat altul dar nu stiu sa stabilesc ordinul de complexitate)
- Stiu sa stabilesc ordinul de complexitate
- Nu stiu nimic despre complexitatea algoritmilor

Trimiteți

Nu trimiteți parole prin formularele Google.

Un produs Google Drive

Acest conținut nu este nici creat, nici aprobat de către Google.
Raportați un abuz - Condiții de utilizare - Condiții suplimentare