Obiectele cu care lucreaza algoritmii si operatii permise

In linii mari, **algoritmii lucreaza cu doua tipuri de obiecte: date si variabile.** Cu acestea sunt permise anumite operatii. **Precizam ca operatiile care vor fi efectuate sunt exemplificate in limbaj de tip pseudocod.**

Date

Asa cum am vazut, orice algoritm porneste de la anumite date de intrare, le prelucreaza, iar in final obtine date de iesire. In exemplul prezentat, datele de intrare sunt reprezentate de cele **10** valori ale lui **x**, iar datele de iesire sunt cele **10** valori ale lui **f(x)**.

Datele pot fi clasificate dupa tipul lor:

- **intregi;**
- **reale;**
- **logice;**
- **sir de caractere.**

Datele din primele doua tipuri poarta numele de **date numerice**.

Datele intregi sunt numere apartinind multimii numerelor intregi. Exemple: **100, -6700, +122**.

Datele reale sunt numere cu zecimale. La o astfel de data, in loc de virgula se foloseste punctul. Exemplu: in loc de **3,25** se scrie **3.25**. Dupa cum stim din matematica, multimea numerelor reale este o reuniune intre multimea numerelor rationale si multimea numerelor irationale. Nici un calculator din lume nu poate retine un numar irational, intrucit acesta are o infinitate de zecimale. Din acest motiv, in informatica, printr-o **data reala** intelegem o valoare cu un numar finit de zecimale.

Datele logice au numai doua valori: **TRUE** (adevarat) si **FALSE** (fals).

Datele de tip sir de caractere sunt reprezentate de siruri de caractere cuprinse intre apostrofuri. Exemple: **'un text'**, **'facem reforma'**.

Aici este momentul sa vorbim de o categorie aparte de date si anume **constantele**. **Constantele sunt date care nu se modifica pe parcursul aplicarii algoritmului.** Acestea pot fi date de oricare dintre tipurile precizate. *Ele se caracterizeaza prin faptul ca sunt utilizate in algoritm, fara a fi citite sau obtinute din calcule.* Se folosesc, de exemplu, in calculele care se fac sau sunt mesaje care trebuie sa apara la fiecare rulare a programului rezultat in urma algoritmului, etc.

Variabile

*O variabila poate retine date **numai** de un tip anume*. Astfel avem variabile care pot retine date intregi, variabile care pot retine date reale, variabile care pot retine date logice si variabile care pot retine date de tip sir de caractere.

De aici rezulta o caracteristica fundamentala a variabilelor: **tipul lor**. Acesta determina natura datelor care pot fi retinute de variabila respectiva. Astfel avem:

- variabile de tip intreg - le vom nota ân pseudocad cu **întreg**;
- variabile de tip real - notate **real;**
- variabile de tip logic -notate **logic;**
- variabile de tip sir - notate **şir.**

Facem urmatoarea conventie: pentru ca un algoritm sa poata folosi o variabila, aceasta trebuie declarata -adica anuntata. Iata cum arata declaratiile variabilelor **a** si **b**:

întreg a
şir b;

Daca trebuie sa declaram mai multe variabile de acelasi tip procedam ca in exemplul urmator:

întreg c, d
logic e, f.

Cu toate ca o variabila are un nume unic, continutul ei poate fi diferit de la un moment la altul, pe parcursul executarii programului. De aici provine denumirea de variabila.

Dar daca, in mod abstract, este posibil sa notam o intrare cu **x** si o iesire cu **f**, cum este posibil sa folosim aceasta abstractizare in momentul in care algoritmul este transpus intr-un limbaj de programare? Practic, in memoria calculatorului se rezerva pentru fiecare variabila un spatiu. Aceasta inseamna ca am rezervat un singur spatiu pentru **x** si un singur spatiu pentru **f**.

Expresii

In scopul efectuarii calculelor, algoritmii folosesc ***expresii***. **O expresie este alcatuita din unul sau mai multi operanzi legati intre ei prin operatori.**

- **Operanzii** pot fi constante si variabile**.**
- **Operatorii** au rolul de a preciza operatiile care se efectueaza.
  - pentru adunare folosim operatorul **'+'**;
  - pentru scadere folosim operatorul '**-**';
  - pentru inmultire folosim operatorul '*****';
  - pentru impartire folosim operatorul '**/**'.

Exemplu: daca **a** si **b** sunt doua variabile de tip intreg care retin **3**, respectiv **7**, atunci expresia **a+3*b** ia valoarea **3+3×7**, adica **24**.

- ✓ In pseudocod putem utiliza ca operator orice simbol cu semnificatie matematica. Exemple: $\sqrt{\ }$ pentru radical, $[\ ]$ pentru parte intreaga etc.

Exemplu: daca **y** este o valoare intreaga care retine **9**, expresia $\sqrt{y}+\left[\frac{y}{2}\right]$ ia valoarea $\sqrt{9}+\left[\frac{9}{2}\right]$, adica $3+[4.5]$, adica **7**.

Operatiile pe care le efectueaza un algoritm

In linii mari, operatiile care se fac intr-un algoritm se impart in trei mari categorii:

- **operatii de intrare / iesire;**
- **operatii de atribuire;**
- **operatii de decizie.**

In trecut, se mai folosea o operatie si anume cea de salt.

Operatii de intrare / iesire

Am aratat ca orice algoritm lucreaza cu **date de intrare** si **iesire.** Acestea pot fi citite sau scrise.

Prin **operatia de intrare** (de citire) se intelege preluarea unei date de la un dispozitiv de intrare catre memoria interna a calculatorului, in zona de memorie rezervata pentru aceasta, adica in variabila. Dispozitivele de intrare pot fi tastatura (cel mai des utilizata); o unitate de discheta; o unitate de disc, etc.

In pseudocod, pentru citire vom folosi operatia **Citeşte**.

Prin **operatia de iesire** (scriere) se intelege preluarea unei date din memoria interna -adica dintr-o variabila- si transferul ei catre un dispozitiv de iesire. Dispozitivele de iesire pot fi monitorul, o unitate de disc, imprimanta, etc.

Pentru scriere vom folosi operatia **Scrie.**

Deocamdata vom presupune ca dispozitivul de intrare este tastatura, iar cel de iesire este monitorul. Sa analizam algoritmul de mai jos -scris in pseudocod- care citeste un numar intreg si il tipareste:

întreg a
Citeşte a
Scrie a

Mai intâi am declarat variabila **a**, de tip intreg -adica are posibilitatea de a retine numere intregi. Urmeaza sa se efectueze operatia de citire a variabilei **a**. Aceasta inseamna ca se asteapta introducerea de la tastatura a datei intregi. In cazul in care introducem numarul **10**, variabila **a** va retine **10**, daca introducem numarul **176,** variabila **a** va retine **176**. Sa presupunem ca am citit **176.** Iata cum ne imaginam variabila **a**:

a

La scriere, pe monitor apare numarul **176** - continutul variabilei **a**.

Observatii

- Dupa scriere, continutul variabilei ramâne nemodificat.

  Fie secventa de mai jos si introducem de la tastatura **17**. Pe monitor apare de doua ori numarul **17**.

  întreg a
  Citeşte a
  Scrie a
  Scrie a

- La o noua citire, continutul vechi al variabilei se pierde.

  Fie secventa de mai jos. Presupunem ca introducem de la tastatura **10, 12** (in aceasta ordine). Atunci variabila **a** va retine **12**, deci pe monitor apare **12.**

  intreg a;
  Citeşte a
  Citeşte a
  Scrie a;

  Se pot citi mai multe variabile cu o singura operatie **Citeşte** si se pot tipari valorile retinute de mai multe variabile cu o singura operatie **Scrie**.

  Exemplu: Fie secventa:

  real a,b,c;
  Citeşte a,b,c
  Scrie a,b,c

  Daca introducem **1.2, -1.3, 12.8,** atunci **a** retine **1.2; b** retine **-1.3**; **c** retine **12.8**. Pe monitor se tipareste **1.2 -1.3 12.8**.

Atribuiri

Prin operatia de atribuire se retine o anumita data intr-o variabila. Ea are mai multe forme, pe care le vom prezenta pe rand.

Forma 1.

v←data

Semnificatia este urmatoarea:

- **v** este numele unei variabile de un tip oarecare.
- ← notatia pentru operatia de atribuire;
- **data** - o valoare de un tip oarecare.

Cu o singura exceptie, tipul variabilei trebuie sa coincida cu tipul valorii atribuite.

Exemple:

intreg a;
a←10;

Variabila **a** va retine **10**.

real b;
b←-7.25
Variabila **b** retine **-7.25**.

şir c;
c←'latina si filosofie'

Variabila **c** retine valoarea de tip sir '**latina si filosofie**'.

Exceptia este urmatoarea: *unei variabile de tip real i se poate atribui o data de tip intreg*:

real d;
d←7

Forma 2

v1←v2

unde, **v1** si **v2** sunt nume de variabile. Cu o exceptie, tipul celor doua variabile trebuie sa coincida. Efectul este ca variabila **v1** va retine continutul variabilei **v2.** Dupa aplicarea acestei operatii, continutul variabilei **v2** ramâne nemodificat, iar continutul initial al variabilei **v1** se pierde.

Exemplul 1

Fie **a** o variabila de tip **intreg** care retine valoarea **2**. Variabila **b** este de acelasi tip cu variabila **a** si retine numarul **3**. Consideram ca se efectueaza atribuirea **a←b**. Continutul initial al celor doua variabile este:

a **b**

Dupa atribuire, continutul celor doua variabile va fi:

a **b**

Exemplul 2.

La fel ca in exemplul 1. Consideram atribuirea **b←a**. Continutul initial al celor doua variabile este:

Dupa atribuire, continutul va fi:

Concluzie. *Nu este indiferent modul de scriere a variabilelor in cadrul atribuirii.* Atribuirea **a←b** nu este identica cu atribuirea **b←a**. Este adevarat ca, dupa atribuire, cele doua variabile vor avea acelasi continut. Insa este important ce continut (al lui **b** dupa prima atribuire, al lui **a** dupa a doua). Aici incepatorii gresesc deseori. *Atentie!*

Forma 3.

v←expresie

Initial se evalueaza expresia, iar valoarea obtinuta este atribuita variabilei **v**. Consideram ca daca cel putin unul din operanzi este real, tipul expresiei este real si poate fi atribuit doar unei variabile de tip real, iar daca toti operanzii sunt intregi, tipul expresiei este intreg si poate fi atribuit unei variabile de tip intreg sau real. Facem conventia ca operatorul '*/*' va da intotdeauna un rezultat real, chiar daca ambii operanzi sunt intregi.

Observatii. Dupa atribuire, variabilele care sunt operanzi ramân nemodificate in ce priveste continutul. Exceptie face, eventual, variabila **v** in cazul in care figureaza si ca operand in expresie. De asemenea, primele doua forme sunt particularizari ale acesteia.

Exemplul 1.

Fie **v** o variabila de tip intreg care contine numarul **7**. Consideram ca se face atribuirea: **v←v+1**. Continutul variabilei **v** inainte de atribuire este:

Dupa atribuire acesta devine:

Explicatie. La pasul 1 a fost evaluata expresia ***v*+1**. In urma evaluarii s-a obtinut valoarea **8** (de tip **intreg**). Aceasta valoare a fost atribuita variabilei **v**. Evident, vechiul continut al lui **v** s-a pierdut.

Exemplul 2.

Fie **c** o variabila de tip **intreg** care retine valoarea **4** si **d** o variabila de tip **real** care retine valoarea **7.3**. Efectuam atribuirea **d←c+1**. Inainte de atribuire cele doua variabile contin:

Dupa atribuire continutul lor va fi:

Explicatie. Am vazut faptul ca este posibil ca unei variabile de tip *real* sa-i atribuim o valoare intreaga. Este normal sa fie asa pentru *ca multimea numerelor intregi este submultime a numerelor reale*.

Operatii de decizie

In pseudocod operatia de decizie are forma urmatoare:

```
Dacă conditie atunci
|   operatie1
| altfel
|  operatie2
|_■
```

Modul de executare este urmatorul:

1. *Se testeaza conditia;*
2. *Daca conditia este indeplinita, se executa* ***operatia$_1$***, *altfel (daca conditia nu este indeplinita) se executa* ***operatia$_2$.***
3. *In continuare se executa operatia aflata dupa operatia decizionala.*

Observatie. Decizia consta in a alege o operatie sau alta spre a fi executata (in nici un caz amândoua). Sa dam un exemplu: *se citesc doua valori intregi a si b. Se cere sa se tipareasca cea mai mare dintre ele.*

```
intreg a,b
Citeşte a
Citeşte b
Dacă a>b  atunci
   |   Scrie a
   | altfel
   |   Scrie b
   |_■
```

In primul rând, se testeaza conditia. In exemplul nostru ea este **a>b.** In cazul in care valoarea retinuta retinut de variabila **a** este mai mare decât cea retinuta de variabila **b,** conditia este indeplinita si se tipareste **a**. Contrar, se tipareste **b.**

Avem posibilitatea ca operatiile subordonate operatiei decizionale sa fie tot operatii decizionale. Exemplu: se citesc **4** valori reale **a, b, c, d.** Sa se evalueze expresia:

$$E=\begin{cases} a+b, & c+d>0 \\ a-b, & c+d=0 \\ a\cdot b & c+d<0 \end{cases}$$

Exemplu numeric. Fie **a=1, b=2, c=3, d=4**. Avem **c+d=7>0**, rezulta ca se va tipari **a+b=1+2=3**. Analizati algoritmul care urmeaza:

```
real    a, b, c, d;
Citeşte a, b, c, d
 Dacă c+d>0  atunci
      |    Scrie a+b
      |  altfel
      |   Dacă c+d=0
      |     |   atunci
      |     |     Scrie a-b
      |     |   altfel
      |     |    Scrie a*b
      |     |_■
      |_■
```

Daca suma **c+d** nu este mai mare decât **0,** rezulta ca este mai mica sau egala cu **0**. Din acest motiv, clauza **altfel** a primului **dacă** va contine o alta operatie decizionala (**dacă)** in urma careia se va decide daca **c+d=0** sau **c+d<0**. In cazul in care **c+d** nu este **0**, singura posibilitate este **c+d<0** (pentru ca deja cunoastem ca nu este mai mare ca **0**). Acesta este motivul pentru care nu mai facem testul **c+d<0**.

✓ Observati faptul ca expresiile se pot calcula si tipari prin utilizarea operatiei **Scrie**.

O problema interesanta este si aceasta: *daca algoritmul impune ca operatia care urmeaza urmeaza lui* **altfel** sa lipseasca**,** *cum procedam?*

Pseudocodul admite o forma particulara a operatiei decizionale.

Dacă conditie **atunci**
| operatie
|_■

Principiul de executare:

1. *Se testeaza conditia;*

2. *In cazul in care aceasta este indeplinita*, *se executa operatia aflata dupa* **atunci***, altfel se trece la operatia urmatoare.*

Exemplu. Se citeste o valoare intreaga. In cazul in care aceasta este para (se imparte exact la **2**) se va afisa mesajul **"am citit un numar par"**. Altfel, programul nu va da mesaj.

```
intreg nr;
Citeşte nr
Dacă ⌊nr/2⌋ = nr/2  -atunci
|   Scrie 'am citit un numar par'
|_■
```

Principiile programării structurate

În general, algoritmii se elaborează prin utilizarea ***exclusivă*** a anumitor structuri (liniară, alternativă, repetitivă) care vor fi prezentate în acest capitol.

Prin structură înțelegem o anumită formă de îmbinare a operațiilor cu care lucrează algoritmii.

Desigur, pentru a fi siguri că acele structuri sunt suficiente pentru elaborarea *oricărui* algoritm au fost necesare anumite demonstrații. Acestea au fost făcute şi nu constituie obiectul actualului curs.

Doresc să lămuresc o anumită problemă. Programarea structurată nu înseamnă doar eliminarea operației de salt **GO TO**. Înseamnă cu mult mai mult, **un mod de gândire a algoritmilor**. Acest mod rezultă din gândirea acestora doar prin structurile permise. **Problema pentru care trebuie să elaborăm algoritmul se descompune după structurile amintite în subprobleme. Apoi acestea se descompun, din nou, după aceleaşi reguli, până se ajunge doar la operații elementare**.

În acest manual vom învăța să redactăm algoritmii direct structurat, varianta nestructurată ținând deja de istoria informaticii.

Structuri de bază, descrierea acestora în pseudocod

Structurile de bază utilizate în descrierea algoritmilor sunt de trei feluri:

- **structura liniară;**
- **structura alternativă;**
- **structura repetitivă.**

Structura liniară

Vom defini structura liniară astfel:

- **Orice operație** (citire / scriere, atribuire, decizională considerată în ansamblul ei) **constituie o structură liniară**;

- **Dacă S1 şi S2 sunt structuri** (de orice tip)**, atunci:**

```
S1
S2
```

este structură liniară.

Pornind de la definiție, se ajunge la următoarea formă de structură liniară:

operația 1
operația 2
.....
operația n.

De ce? Operația **1** este structură liniară, operația **2** este structură liniară, rezultă că operația **1**, urmată de operația **2** este structură liniară şi cum operația **3** este structură liniară, înseamnă că operația **1** urmată de operația **2**, urmată de operația **3** este structură liniară ş.a.m.d. Ordinea de executare este: operația **1**, apoi operația **2**, ..., până la operația **n.**

Exemplul 1. *Se citesc două variabile reale. Să se interschimbe conținutul lor şi să se tipărească.*

```
real a, b, man;
Citeşte a, b
man ← a;
```

```
a ←b;
b ← man;
Scrie a, b
```

- se citesc cele două variabile -operația **1**, de citire;
- variabilei **man** i se atribuie conținutul variabilei **a**- operația **2**, de atribuire;
- variabilei **a** i se atribuie conținutul variabilei **b**- operația **3**, de atribuire;
- variabilei **b** i se atribuie conținutul variabilei **man**- operația **4**, de atribuire;
- se afişează **a**, **b** -operația **5**.

Structura alternativă

Structura alternativă se defineşte astfel:

➢ **Dacă S1 şi S2 sunt structuri şi E este o condiție, atunci:**

```
Dacă E atunci
     |     S1
     |  altfel
     |     S2
     |_■
```

este structură alternativă.

Mecanismul de executare este:

- se evaluează condiția;
- dacă aceasta este îndeplinită se execută **S1**, altfel se execută **S2**.

Forma simplificată a structurii alternative este

➢ **Dacă S este o structură şi E este o condiție, atunci:**

```
Dacă E atunci
     |     S1
     |_■
```

este structură alternativă.

Dacă la evaluare **E** este îndeplinită se execută **S**, altfel se trece mai departe.

Exemplul 1. *Să se scrie un algoritm care citeşte un număr natural (comanda). Dacă acesta este 0 se vor citi două numere întregi a şi b şi se va tipări suma lor, contrar se vor citi două numere reale x şi y şi se va tipări suma lor.*

În ambele cazuri trebuie executate mai multe instrucțiuni. Pentru aceasta am folosit structura alternativă.

```
intreg comanda, a, b, s1;
real  x, y, s2;
Citeşte comanda
Dacă comanda = 0  atunci
     |     Citeşte a,b
     |     s1 ←a+b
     |     Scrie s1
     |   altfel
     |    Citeşte x, y
     |    s2 ←x+y
     |    Scrie s2
     |■
```

```
Citeşte  a,b          Citeşte  x,y
s1 ←a+b               s2 ←a+b
Scrie s1              Scrie s2
    ↑                     ↑
   P1                    P2
```

Cele două secvențe de mai sus sunt structuri liniare. Le vom nota cu P_1 şi P_2.

În aceste condiții algoritmul în pseudocod devine:

```
intreg comanda, a, b, s1;
real  x, y, s2;
Citeşte comanda
Dacă comanda = 0  atunci
    |     P1
    |   altfel
    |     P2
    |■
```

Structura alternativă, în asamblul ei, o vom nota cu **A**. În aceste condiții algoritmul devine:

```
intreg comanda, a, b, s1
real  x, y, s2;
Citeşte comanda
A
```

Cele două structuri (facem abstracție de declarații) alcătuiesc o structură liniară. pe care o notăm cu **P**.

În aceste condiții "rezolvarea" este:

Declarații de variabile

P

După cum observați, totul s-a redus la o singură structură prin respectarea regulilor impuse de programarea structurată.

În realitate, dv. veți proceda invers atunci când scrieți un algoritm: prin respectarea regulilor programării structurate, descompuneți problema în subprobleme, pe acestea le descompuneți în mod asemenător, până când ați obținut doar operații elementare.

Exemplul 2. Se citesc **a** şi **b**, numere reale. Să se rezolve ecuația de gradul 1 $ax + b = 0$.

În matematică, ecuația de gradul **1** este definită numai dacă **a≠0**. În informatică, trebuie tratat şi cazul **a=0**, pentru că nimeni nu garantează că utilizatorul nu greşeşte la introducerea datelor.

Cum gândim? Într-o primă etapă algoritmul nostru va testa dacă **a** este diferit de **0**. În caz afirmativ, se va rezolva ecuația în mod clasic, contrar se va trata excepția. Atât **Rezolvă ecuația** cât şi **Tratează excepția** sunt structuri pe care le vom trata în etapa următoare.

```
real a,b,x
Citeşte a,b
Dacă a≠0 atunci
  |   Rezolvă ecuația
  | altfel
  |   Tratează excepția
  |■
```

Rezolvarea ecuației presupune calculul şi afişarea rădăcinii (ambele sunt operații elementare), iar dacă **a=0**, rămâne să vedem cum este **b** pentru a determna dacă ecuația nu admite soluții (**b≠0**), sau admite o infinitate de soluții (**b=0**).

Iată algoritmul:

```
real a,b,x
Citeşte a,b
Dacă a≠0 atunci
  |     x ← -b/a
  |     Scrie x
  | altfel
  |   Dacă b≠0 atunci
  |     |    Scrie " Ecuația nu admite soluții"
  |     | altfel
  |     |    Scrie " Ecuația admite o infinitate de soluții"
  |     |■
  |■
```

- ✓ În vreme ce operația decizională subordonează sub **dacă** sau sub **altfel** câte o singură **operație**, structura alternativă subordonează sub **dacă** sau sub **altfel** câte o singură **structură**! Cu alte cuvinte, structura alternativă generalizează operația decizională.

Structura repetitivă

Structura repetitivă apare în următoarele variante:

- **condiționată anterior -**este de două feluri:
  - **Cât timp... execută (While Do)**
  - **Pentru... execută (For)**
- **condiționată posterior** -
  - **Repetă ... până când (Repeat until)**
  - **Execută ... cât timp (Do While);**

Singura structură indispensabilă este **Cât timp... execută (While Do)** - restul se pot obține din aceasta.

Structura Cât timp... execută (While Do)

Structura de tip **Cât timp... execută** se defineşte astfel:

- **Dacă E este condiție şi S o structură atunci**

```
Cât timp E execută
  | S
  |■
este o structură de tip  Cât timp ... execută
```

Principiul de executare este următorul:

P1 Se evaluează condiția. Dacă este îndeplinită, se execută **S** şi se trece la **P2**, altfel executarea se încheie;
P2. Se reia pasul **P1**.

Observație. Se observă faptul că **S** se execută **numai** dacă condiția este îndeplinită. Din acest motiv, structura se mai numeşte condiționată anterior. De la bun început precizăm următoarele:

⇒ este obligatoriu să folosim structura **Cât timp... execută** atunci când sunt îndeplinite simultan două condiții:

- instrucțiunea subordonată se execută (de câte ori este cazul) numai dacă este îndeplinită o anumită condiție;
- în momentul în care se intră în secvența repetitivă nu se ştie de câte ori aceasta se va executa.

Exemplul 1. *Se citeşte un număr natural* **n***, diferit de* **0***. Să se tipărească suma cifrelor sale.* Dacă se citeşte **248** se va tipări **14**.

```
intreg i, n, s;
Citeşte n
s ← 0
Cât timp  n <> 0 execută
  |  s  ←  s+n mod 10;
  |  n  ←  n div 10
  |■
Scrie s
```

Exemplu numeric. Se citeşte **n = 248.**

248≠0, 248 mod 10 = 8, s=0+8=8; n=24;
24≠0, 24 mod 10 = 4, s=8+4=12; n=2;
2≠0, 2 div 10 = 0, s=12+2=14; n=0;
0=0, se tipăreşte **s=14.**

Structura de tip Pentru...execută

➢ **Fie i o variabilă de tip intreg, numită variabilă de ciclare, a si b două valori intregi (sau două variabile de tip intreg) numite *valoare inițială* şi *valoare finală*, şi S o structură. Atunci:**

```
Pentru i ←a,b execută
  | S
  |■
```

este o structură de tip Pentru ...execută

Principiul de executare este următorul:

P1. Variabila de ciclare **i** ia valoarea inițială **a**;
P2. Dacă **i** este mai mic sau egal cu **b**, se execută **S**, se adună **1** la **i** şi se reia **P2**. Altfel, executarea este încheiată.

Această structură poate fi simulată cu **Cât timp... execută**
astfel:

```
i ← a;
Cât timp i <= b execută
    |  S
    |  i ← i+1
    |■
```

Se utilizează structura **Pentru...execută** dacă:

- secvență se repetă;
- se cunoaşte, înaintea intrării în secvență, de câte ori trebuie să fie executată secvența.

Exemplul 1. *Se citeşte **n** (număr natural). Se cere să se efectueze suma primelor **n** numere naturale. Exemplu:* **n=3**. **S=1+2+3=6**.

Cum procedăm? Este posibil să cunoaşteți formula care dă suma primelor **n** numere naturale:

$$S_n = \frac{n \cdot (n + 1)}{2}.$$

Dacă aplicăm această formulă problema devine foarte uşoară. Întrucât exemplul este dat pentru a învăța structura **Pentru...execută**, presupunem că nu o cunoaştem. Vom aduna, pe rând, numerele **1, 2,...,n** la conținutul unei variabile **s**. Inițal, **s** reține valoarea **0**. Cum gândim? Din start, avem de adunat **n** numere. Deci, vom avea **n** paşi (la fiecare pas se adună un număr). Să notăm cu **i** numărul care se adună la **s.** Instrucțiunea de atribuire va fi: **s ← s+i.** Prima dată **i** va fi **1**. A doua oară **i** va fi **2** ş.a.m.d.

Avem:

i←1, s ← s+1; s=0+1=1;
i←2, s ← s+2; s=1+2=3;
i←3, s ← s+3; s=3+3=6;
i←4, s←s+4; s=6+4=10.
...

Prezentăm algoritmul obținut:

```
intreg n, i, s;
read n
s ← 0
Pentru i ← 1, n execută
    | s ← s+i
    |■
Scrie s
```

Structura de tip Repetă...până când

➢ **Fie S o structură şi E o condiție. Atunci:**

```
Repetă
   | S
   |■
  până când E
```

este o structura Repetă .. .până când

Principiul de executare este următorul:

P1 Se execută **S**;
P2 Dacă **E** este **nu** este îndeplinită, se trece la **P1**, altfel executarea este încheiată.

Observați faptul că **S** se execută întotdeauna o dată. Apoi se testează faptul dacă se mai execută sau nu. {i această structură poate fi simulată cu ajutorul structurii **Cât timp ... execută**

Iată cum:

```
S;
Cât timp not E execută
 |  S
 |■
```

Iată cum se poate calcula suma primelor **n** (**n** ≥ 1) numere naturale prin utilizarea acestei structuri:

```
intreg n, i, s;
Citeşte n
i ← 1
s ← 0
 Repetă
  |  s ← s+i
  |  i ← i+1
  |■
 până când i>n
 Scrie s
```

Să vedem cum decurge algoritmul după citirea lui **n** (citim **3**):

- **i** ia valoarea **1,** iar **s** ia valoarea **0**;
- **s** ia valoarea **0+1=1**, iar **i** ia valoarea **2**;
- **i** nu este mai mare decât **3**, deci **s** ia valoarea **1+2=3**, iar **i** va lua valoarea **3**;
- **i** nu este mai mare ca **3**, deci **s** va lua valoarea **3+3=6**, iar **i** va lua valoarea **4**;
- **i** este mai mare decât **n**, deci se trece la instrucțiunea următoare, unde se tipăreşte valoarea **6**.

✓ O astfel de structură se utilizează în limbajul **Pascal**.

Structura Repetă ... cât timp

➢ **Fie S o structură şi E o condiție. Atunci:**

```
Repetă
 | S
 |■
cât timp E
```

este o structura Repetă .. .cât timp.

Acesată structură are acelaşi mecanism de executare ca **Repetă până când,** diferența fiind dată de faptul că secvența **S** se execută din nou dacă **E** este îndeplinită.

✓ O astfel de structură se utilizează în limbajul **C++**.