Iterativitatea și recursivitatea

Tehnici de elaborare a algoritmilor

Ciobanu bogdaN  clasa a 11-a ‘’C’’

2020

**Cuprins:**

[1. Introducerea 2](#_Toc37877850)

[2. Iterativitatea 2](#_Toc37877851)

[2.1 Instrucțiunea for 3](#_Toc37877852)

[2.2 Instrucțiunea while 4](#_Toc37877853)

[2.3 Instrucțiunea repeat 4](#_Toc37877854)

[2.4 Avantaje și dezavantaje 5](#_Toc37877855)

[2.5 Exemple de programe 5](#_Toc37877856)

[3.Recursivitate 8](#_Toc37877857)

[3.1 Avantaje și dezavantaje 9](#_Toc37877858)

[3.2 Exemple de programe 9](#_Toc37877859)

[4. Concluzie 11](#_Toc37877860)

# **1.** **Introducerea**

Algoritm Un algoritm (cuvântul are la origine numele matematicianului persan Al-Khwarizmi) înseamnă în matematică şi informatică o metodă sau o procedură de calcul, alcătuită din paşii elementari necesari pentru rezolvarea unei probleme sau categorii de probleme. De obicei algoritmii se implementează în mod concret prin programarea adecvată a unui calculator, sau a mai multora. Clasificări În funcţie de modul de implementare, un algoritm poate fi: recursiv- face uz de sine însuşi, în mod iterativ (repetitiv).

## **2. Iterativitatea**

Iterația reprezintă repetarea unei secvențe de instrucțiuni de un număr specificat de ori sau până la îndeplinirea unei condiții [6]. Mecanismul dat este utilizat atunci când este necesară realizarea unei anumite acțiuni de mai multe ori pe baza anumitor condiții. Mecanismul clasic de construcție a iterațiilor într-un cod de programare se face prin așa-numitele cicluri. Algoritmii iterativi se opresc când condiția ciclului devine falsă. Un ciclu poate fi infinit dacă condiția de continuare a ciclului este mereu adevărată. Pascalul prezintă următoarele structuri iterative: ciclul for, ciclul while… do și ciclul repeat... until.

## **2.1 Instrucțiunea for**

Instrucţiunea for indică execuţia repetată a unei instrucţiuni, care poate fi și compusă, în funcţie de valoarea unei variabile de control. Sintaxa instrucţiunii în studiu este:

<Instrucțiunea **for**>::= **for** <Variabilă> := <Expresie> <Pas> <Expresie> **do** < Instrucțiune> ; <Pas> ::= to | downto.

Variabila situată după cuvântul-cheie **for** se numește **variabilă de control** sau **contor**. Această variabilă trebuie să fie de tip ordinal. Valorile expresiilor din componența instrucțiunii **for** trebuie să fie compatibile, în aspectul atribuirii, cu tipul variabilei de control. Instrucțiunea situată după cuântul-cheie **do** se execută pentru fiecare valoare din domeniul determinat de valoarea inițială și de valoarea finală.

Dacă instrucțiunea **for** utilizează pasul **to**, valorile variabilei de control sunt incrementate la fiecare repetiție, adică se trece la succesorul valorii curente. Dacă valoarea inițială este mai mare decât valoarea finală, instrucțiunea situată după cuvântul-cheie **do** nu se execută niciodată. Dacă instrucțiunea **for** utilizează pasul **downto**, valorile variabilei de control sînt decrementate la fiecare repetiție, adică se trece la predecesorul valorii curente. Dacă valoarea inițială este mai mică decât valoarea finală, instrucțiunea situată după cuvântul-cheie **do** nu se execută niciodată.

Valorile variabilei de control nu pot fi modificate în interiorul ciclului.

## **2.2 Instrucțiunea while**

Instrucțiunea **while** conține o expresie booleană care controlează execuția repetată a altei instrucțiuni, care poate fi și una compusă. Sintaxa instrucțiunii în studiu este:

<Instrucțiunea **while**> ::= **while** <Expresie booleană> **do** <Instrucțiune>.

Instrucțiunea situată după cuvântul-cheie **do** se execută repetat atâta timp, cât valoarea expresiei booleene este true. Dacă expresia booleană ia valoarea false, instrucțiunea de după **do** nu se mai execută. Instrucţiunea **while** se consideră deosebit de utilă în situaţia în care numărul de execuţii repetate ale unei secvenţe de instrucţiuni este dificil de evaluat.

## **2.3 Instrucțiunea repeat**

**repeat** indică repetarea unei secvențe de instrucțiuni în funcție de valoarea unei expresii booleene. Sintaxa instrucțiunii este:

**<** Instrucțiunea **repeat>** ::= **repeat** < Instrucțiune> {;<Instrucțiune>} until <Expresie booleană>

Instrucţiunile situate între cuvintele-cheie **repeat** și **until** se execută repetat atât timp cât expresia booleană este falsă. Când această expresie devine adevărată, se trece la instrucțiunea următoare. Instrucțiunile aflate între **repeat** și **until** vor fi executate cel puțin o dată, deoarece evaluarea expresiei logice are loc după ce s-a executat această secvență.

În mod obișnuit, instrucțiunea **repeat** se utilizează în locul instrucțiunii **while** atunci când evaluarea expresiei care controlează repetiția se face după executarea secvenței de repetat. Aceasta este utilă în situaţia în care numărul de executări repetate ale unei secvenţe de instrucţiuni este dificil de evaluat.

## **2.4 Avantaje și dezavantaje**

Avantaje:

• Necesarul de memorie este mai mic decât în cazul recursivității;

• Testarea și depanarea programelor este simplă;

• Un algoritm iterativ poate fi mai rapid decât echivalentul său recursiv, din cauza eforturilor adiționale, precum apelarea funcțiilor și înregistrarea stivelor în mod repetat, prezente în cazul algoritmul recursiv.

Dezavantaje:

##### • Structura programului poate fi complicată;

• Volumul de muncă necesar pentru scrierea programului poate fi mare.

## **2.5 Exemple de programe**

1. ***Calcularea sumei numerelor de la 1 până la N.***

**program** p1;

**var** n,sum,i:integer;

**begin**

readln(n);

**for** i:=1 **to** n **do begin** {Adunam numerele de la 1 la **N** pentru a afla}

sum:=sum+i; {suma numerelor}

**end**;

writeln(sum);

**end**.

1. ***Calcularea sumei pătratelor numerelor de la 1 la N.***

program p2;

var n,i:integer;

sum:longint;

begin

readln(n);

for i:=1 to n do begin {Adaugam la suma patratele nr-lor de la 1 la N}

sum:=sum+(i\*i);

end;

writeln(sum);

end.

1. ***Calcularea produsului numerelor de la 1 la N.***

program p3;

var n,i:integer;

produs:longint;

begin readln(n);

produs:=1;

for i:=1 to n do begin produs:=produs\*i; {Inmultim numerele de la 1 la N}

end;

writeln(produs);

***4.Calcularea sumei numerelor pare și a celor impare de la 1 la N.***

program p4;

var n,i:integer;

sum\_p,sum\_i:integer; begin readln(n);

for i:=1 to n do begin {De la 1 la N} if i mod 2 = 0 then sum\_p:=sum\_p+i else {Determinam daca nr e par/impar}

sum\_i:=sum\_i+i;

{Adaugam nr-ul la suma respectiva} end;

writeln('suma nr-lor pare : ',sum\_p);

writeln('suma nr-lor impare : ',sum\_i);

end.

***5.Calcularea sumei numerelor de la 1 la N ce sunt divizibile la numărul X.***

program p5;

var x,n,i,sum:integer;

begin write('limit : ');

readln(n); write('divisor : ');

readln(x); {divizorul}

for i:=1 to n do begin

if i mod x = 0 then begin {Daca i e divizibil la X atunci}

sum:=sum+i; {Suma multiplilor se mareste cu valoarea lui i}

end;

end;

writeln(sum);

end.

### **3.Recursivitate**

În matematică şi informatică, recursivitatea sau recursia este un mod de a defini unele funcţii. Funcţia este recursivă, dacă definiţia ei foloseşte o referire la ea însăşi, creând la prima vedere un cerc vicios, care însă este numai aparent, nu şi real.

Recursivitatea e strins legata de iteratie, dar daca iteratia e executia repetata a unei portiuni de program, pana la indeplinirea unei conditii (while, repeat, for), recursivitatea presupune executia repetata a unui modul, insa in cursul executiei lui (si nu la sfirsit, ca in cazul iteratiei), se verifica o conditie a carei nesatisfacere, implica reluarea executiei modulului de la inceputul sau. Atunci un program recursiv poate fi exprimat: P=M(Si,P) , unde M este multimea ce contine instructiunile Si si pe P insusi. Structurile de program necesare si suficiente in exprimarea recursivitatii sint procedurile si subrutinele ce pot fi apelate prin nume. In PASCAL, exista doua tipuri de parametri formali (ce apar in antetul unei proceduri sau functii): valoare si variabila (ultimii au numele precedat de cuvintul cheie var).

Exista două tipuri de recursivitate:

1) recursivitate directă - cand un subprogram se autoapelează în corpul său ;

2) recursivitate indirectă - cînd avem două subprograme (x si y), iar x face apel la y şi invers ;

Se folosesc algoritmi recursivi atunci cînd calculele aferente sunt descrise în forma recursive.

Recursivitatea este frecvent folosită în prelucrarea structurilor de date definite recursiv. Un subprogram recursiv trebuie scris astfel încat să respecte regulile :

a) Subprogramul trebuie să poată fi executat cel puţin o dată fără a se autoapela;

b)Subprogramul recursiv se va autoapela într-un mod în care se tinde spre ajungerea în situaţia de execuţie fără autoapel.

### **3.1 Avantaje și dezavantaje**

Avantaje:

-se realizează programe mai rapide;

-se evită operaţiile mult prea dese de;

salvare pe stiva calculatorului;

-se evită incărcarea calculatorului în

cazul apelurilor repetate.

Dezavantaje:

-în cazul unui nr. mare de autoapelări, există posibilitatea ca segmentul de stiva să depaşească spaţiul alocat, caz în care programul se va termina cu eroare;

-recursivitatea presupune mai multă memorie în comparaţie cu iterativitatea.

### **3.2 Exemple de programe**

***1.Calcularea sumei numerelor de la 1 până la N.***

function sum(n:integer):integer;

begin

if n=1 then sum:=1 else begin

sum:=n+sum(n-1); {Adaugam N la suma, apoi reapelam}

end; {f-ctia cu N-1, adaugand nr-ul la}

end; {suma, repetam procesul pana N=1}

***2.Calcularea produsului numerelor de la 1 la N.***

function produs(n:integer):longint;

begin

if n=1 then produs:=1 else begin {Inmultim produsul{cu valoarea 1}la N} produs:=n\*produs(n-1); {Reapelam f-ctia cu parametrul N-1}

end; {Inmultind produsul}

end;

***3.Calcularea sumei pătratelor numerelor de la 1 la N.***

function suma\_patratelor(n:integer):longint;

begin

if n=1 then suma\_patratelor:=1 else begin {Adaugam la suma patratul}

suma\_patratelor:=n\*n+suma\_patratelor(n-1);{numerelor de la N la 1}

end;

end;

***4.Calcularea sumei numerelor pare și a celor impare de la 1 la N.***

function sum(n:integer; var sum\_i,sum\_p:longint):longint; begin if n=1 then begin

sum\_i:=sum\_i+1;

end else begin {In incinta f-ctie determinam}

if n mod 2 = 0 then sum\_p:=sum\_p+n else {daca N e par/impar}

sum\_i:=sum\_i+n; {Adaugam nr la suma respectiva} sum(n-1,sum\_i,sum\_p); {Reapelam f-ctia cu parametru N-1} end;

end;

***5.Calcularea sumei numerelor de la 1 la N ce sunt divizibile la numărul X.***

procedure sums(x:integer; divisor:integer; var sum:integer); begin

if x=0 then sum:=sum+0 else begin

if x mod divisor = 0 then begin

sum:=sum+x; {Daca X se imparte exact la divizor} writeln(x); {atunci prodecure se auto-apeleaza}

sums(x-1,divisor,sum); {cu valoarea lui X scazuta cu 1} end else sums(x-1,divisor,sum);

end;

end;

#### 4. Concluzie

Ambele tehnici sunt utile și importante oentru programare. Înainte de a începe un program, trebuie mereu să ne alegem una din aceste variante, ca să avem un rezultat mai bun și mai productiv, iar iterativitatea este ca să facem programe mai simple și mai scurte, iar dacă programul necesită mai multe condiții cu o dificultate mai grea, atunci folosim recursivitatea.

5. Bibliografie

https://prezi.com/qfmfcl\_7jdpg/recursivitate-si-iterativitate/

https://prezi.com/hwzqekzxc5o9/iterativitate-sau-recursivitate/

https://www.codeit-project.eu/ro/differences-between-iterative-and-recursive-algorithms/