# Tehnici de elaborare a algoritmilor

Coșciug Andrian, cl XI-a ‘D’

# CUPRINS

## Cuprins

Descrierea metodei /Metode teoretice ..... 3
Iterativitate ..... 3
Exemple de programe iterative ..... 4
Recursivitate ..... 6
Probleme recursive ..... 7
Concluzie ..... 8
Bibliografie ..... 8

## Descrierea metodei /Metode teoretice

### Iterativitate

Iterația folosește instrucțiuni de buclă , pînă cînd programul nu va da o eroare și va afișa la ecran rezultatul final.

Bucla iterativă este instrucțiunea care pune programul să treacă autmat la celula următoare pînă cănd nu obține o valoare admisibilă care poate fi utilizată pe tot parcursul programului

În comparație cu metoda recursivă, metoda iterativă este mai ușor de aplicat , iar un program poate fi transformat mai uşr din recursiv în iterativ , decît din iteratic în recursiv. În cazul în care un program este scris în mod analog în mod iterativ și recursiv, programul iterativ va rula mai repede din cauza cheltuielilor generale care pot fi funcțiile de apel și stivele de înregistrare dar nuse limitează la aceste 2 funcți.

În abordarea iterativă condiția se repetă pînăcînd funcția nu reușește , iterația se termină atunci cînd condiția pricipală a programului eșuează , iar dacă testul de eșuare nudevine fals atunci programul de iterațieva rula la infinit.

În programele iterative există 3 tipuri de bucle

1. Bucla cu parametri (for..to..do)
2. Bucla cu precondiții(while..do)
3. Bucla cu o postcondiție (do..while)

## Exemple de programe iterative

1. Cod iterativ ce calculează factorialul f

```
Program iterativ
var i,n:integer; f:real;
function f(n:integer):real; ;
begin
S:=0
for i:=1 to n do  f:=f*i;
end;
end.
```

2. program iterativ ce calculează următoarea sumă  S=2+4+6+…+200

```
Program iterativ
Var i,s:real;
Function it(i:real):real;
s := 0;
i := 0;
while i<200 do
begin
 i:=i+2;
 s:=s+i;
end;
```

3 program iterativ care calculează suma S(n) = 1 + 3 + 5 + ... + (2n – 1)

```
program iterativ;
```

```
var m: integer;
function rec(n:integer):real;   i:integer , s:real;
begin
S:=0
For i:= 1 to n do
S:=S+(2*i-1)
S1:=S;
End;
End.
```

4. program iterativ care calculeaza  suma S=1/3+3/5+5/7+…+(2n-1)(2n+1)

```
program iterativ;
var m:integer;
Function S1(n:integer):real;   i:integer , s:real;
Begin
S:=0
For i:=1to n do
S:=S+(2*i-1)/(2*i+1);
S1:=S;
End;
End.
```

5. program iterativ ce calculează S=2*2+3*4+4*6+n*(2*(n-1))

```
 Program iterativ;
  Var n:integer;
    Function suma (n:integer):integer;
Var i,s:integer;
Begin
S:=0
For I :=1 to n do
S:=S+i*(2*(i-1))
Suma:=S
End;
```

## Recursivitate

Apelul unui program recursiv , ca si al unuia nerecursiv presupune alocarea de memorie pe stiva variabilelor locale, a parametrilor si a adresei de revenire.In cazul utilizarii recursivitatii se mai recomanda utilizarea variabilelor globale pentru a evita crearea unor dubluri de variabile pe stiva. Avantajul recursivitatii este acela ca:

-Reduce lungimea textului sursa

-Conduce la solutii clare in comparatie cu solutiile iterative

Recursivitatea se recomanda:

-În probleme a caror solutii pot fi definite in termeni recursivi

-În cazul problemelor complexe rezolvate prin backtracking

Întotdeauna o problema recursiva are si o solutie iterativa si invers.

Recursivitatea poate fi directa atunci cand in corpul subprogramului apare apelul acelui subprogram , sau indirecta daca apelul apare in corpul altui subprogram care se apeleaza din primul subprogram.Autoapelul provoaca o noua activare a aceluiasi subprogram , ceea ce presupune executia instructiunilor subprogramului, incepand cu prima instructiune a acestuia si pana la auto apel , cand se activeaza din nou subprogramul. Pentru a evita o asemenea situatie trebuie ca autoapelul programului sa fie conditionat de indeplinirea unei anumite conditii. Daca conditia de autoapel (C) este indeplinita , atunci subprogramul este reactivat (reapelat) , in caz contrar sirul de activari (apeluri) ale subprogramului se intrerupe si se executa secventele ramase de executat din subprogram, activate in ordinea inversa activarii.

Recursivitatea este un mecanism general de elaborare a programelor.

Recursivitatea s-a utilizat initial pentru transcrierea formulelor matematice descrise recursiv, ulterior extinzandu-se pentru elaborarea multor algoritmi.

Spunem ca o notiune este definita recursiv daca in cadrul definitiei intervine insasi notiunea care se defineste.Intalnim procese care se exprima in termeni recursivi, sau in matematica apar frecvent relatii recursive, denumite si relatii de recurenta.

In programare, instrumental necesar si suficient pentru a exprima programe recursive este subprogramul pentru ca subprogramul da un nume unei actiuni care paote fi invocata prin numele respectiv

## Probleme recursive

1.program recursiv care calculeaza  suma S=1/3+3/5+5/7+…+(2n-1)(2n+1)

```
program recursie;
var m:integer;
function s(n:integer):real;
Begin
If n=0 then s:=0
Else s:=s(n-1)+(2*n-1)/(2*n-1);
End;
End.
```

2. program recursiv ce calculează S=2*2+3*4+4*6+n*(2*(n-1))

```
Program recursiv;
 Var m:integer;
Function suma (n:integer):integer;
Begin
If n=0 then suma:=0
Else suma :=suma(n-1)+n*(2*(n-1))
End;
End.
```

3.  a*b prin adunare repetată

```
function produs(a,b:word):word;
var p:word;
begin
p:=0;
while b<>0 do begin
 p:=p + a;
 b:=b – 1;
end;
produs:=p end;
```

4. program recursiv care calculează suma S(n) = 1 + 3 + 5 + ... + (2n – 1)

```
program recursie;
 var n: integer;
 function rec(m:integer):integer;
 begin
if m=1 then rec:=1
 else rec:=rec(m-1)+(2*m-1);
end;
begin
```

```
  write('n= '); read(n);
  writeln ('suma= ', rec(n));
end.
```

5 Elaborati o functie recursiva ce calculeaza sirul : S= 18 +1/3+1/4+1/5+1/n

```
program SumaCuRecursie;
var n:integer;
function suma(m:integer):real;
begin
  if m<3 then suma:=18
  else suma:=suma(m-1) + 1/m;
end;
begin
 write('n='); read(n);
 writeln('suma 18+ 1/3 + 1/4 + 1/5 + ... +1/n = ', suma(n));
end.
```

## Concluzie

În concluzie putem afirma că programele recursive necesită mai multă memorie decît cele iterative , însă timpul a unor programe iterative și recursive sunt analoage , dar structurarea programului este mult mai complicată în programele iterative, și respectiv volumul de muncă pentru a scrie un program iterativ este mai mare decît munca depusă în programul recursiv. Testarea și elaborarea modificărilor este mai simplă în cadrul programelor iterative care pot fi modificate simplu în cadrul efectuării. Programele iterative necesita mai putina memorie decat cele recursive, iar astfel ele pot fi procesate de calculator mai repede .

## Bibliografie


1. http://www.authorstream.com/Presentation/aSGuest41792-360322-iterativitate-sau-recursivitate-rodika-guzun-science-technology-ppt-powerpoint/
2. https://prezi.com/hwzqekzxc5o9/iterativitate-sau-recursivitate/
3. https://www.slideshare.net/Vytamin/iterativitate-sau-recursivitate
4. http://staff.cs.upt.ro/~ioana/sdaa/sda/l1.html
5. https://innf.weebly.com/recursivitate.html
6. https://www.scribd.com/doc/98652268/iterativitatesirecursivitate