



ITERATIVITATE SAU RECURSIVITATE

Cuprins

1. Aspecte Teoretice	2
1.1.Iterativitatea	2
1.2. Recursivitatea	2
Tipuri de recursii.....	2
1.3.Observații importante	3
2.Probleme rezolvate	4
2.1. Exemple iterative.....	4
2.1.1. Determinarea minimului de pe pozițiile impare ale unui vector.....	4
2.1.2. Media Aritmetica a elementelor unul vector, prin funcție.....	5
2.1.3. Suma elementelor pare de pe pozițiile impare.....	5
2.1.4. Inlocuirea lui DA cu Nu.....	6
2.1.5. Stergerea elementelor dintr-un string	7
2.2. Exemple Recursive.....	7
2.2.1. Fibonacci	7
2.2.2. Suma unui sir	8
2.2.3. Arbore binar	9
2.2.4 Produsul elementelor.....	10
2.2.5. Reversarea unui string	11
3.Diferența dintre iterativitate și recursivitate.	11
Bibliografie	13

1. Aspecte Teoretice

1.1.Iterativitatea

Definiție

Iterativitatea este procesul prin care rezultatul este obținut ca urmare a execuției repetate a unui set de operații, de fiecare dată cu alte valori de intrare. Numărul de iterații poate fi necunoscut sau cunoscut, dar determinabil pe parcursul execuției. Metoda de repetitivitate este cunoscută sub numele de ciclu (loop) și poate fi realizată prin utilizarea următoarelor structuri repetitive: ciclul cu test inițial, ciclul cu test final, ciclul cu număr finit de pași. Indiferent ce fel de structură iterativă se folosește este necesar ca numărul de iterații să fie finit.

De exemplu, procesul simplu de numărare, din 1 în 1, până la 100, este condus **iterativ**, punându-se în evidență funcția matematică de succesiune definită pe mulțimea numerelor naturale. [1]

1.2. Recursivitatea

Una din cele mai răspândite tehnici de programare este recursia.

Recursia se definește ca o situație în care un subprogram se autoapelează fie direct, fie prin intermediul altui subprogram.

În matematică și informatică, **recursivitatea** sau recursia este un mod de a defini unele funcții. Funcția este recursivă, dacă definiția ei folosește o referire la ea însăși, creând la prima vedere un cerc vicios, care însă este numai aparent, nu și real.

Tipuri de recursii

Există două forme de recursivitate:

- recursivitatea directă, care se prezintă prin autoapelul subprogramului în interiorul lui;
- recursivitatea indirectă, în care un subprogram apelează un alt subprogram, iar acesta îl apelează pe primul. Procesul se încheie în momentul în care este îndeplinită condiția de ieșire din subprogramul care a inițiat procesul de apeluri reciproce.[3]

Exemple:

- factorialul unui număr: $N! = N \cdot (N-1)!$ $N! = N \cdot (N-1)!$;
- ridicarea la putere: $a^n = a \cdot a^{n-1}$ $a^n = a \cdot a^{n-1}$;
- termenul unei progresii aritmetice: $a_n = a_{n-1} + r$ $a_n = a_{n-1} + r$;
- șirul lui Fibonacci: $F_n = F_{n-1} + F_{n-2}$ $F_n = F_{n-1} + F_{n-2}$;
- etc. [5]

1.3.Observații importante

- Orice funcție recursivă trebuie să conțină o condiție de reluare a apelului recursive (sau de oprire).
- La fiecare reapel al funcției se execută aceeași secvență de instrucțiuni.
- Ținând cont de observațiile anterioare, pentru a implementa o funcție recursivă, trebuie să:
 1. Identificăm relația de recurență (ceea ce se execută la un moment dat și se reia la fiecare reapel)
 2. Identificăm condițiile de oprire ale reapelului .
- În cazul în care funcția are parametri, aceștia se memorează ca și variabilele locale pe stiva, astfel:
 1. Parametrii transmiși prin valoare se memorează pe stivă cu valoarea din acel moment
 2. Pentru parametri transmiși prin referință se memorează adresa lor.[4]

2.Probleme rezolvate

2.1. Exemple iterative

2.1.1. Determinarea minimului de pe pozitiile impare ale unui vector.

Type Vector=array [1..100] of integer;

Var A,B: vector;

i ,n: integer;

minA, minB: integer;

Function Minim(T, V:vector; j:integer);

Begin

If T[j]>V[j] **then** Minim:= V[j]

Else Minim:=T[j];

End;

BEGIN

Writeln('n='); read(n);

For i:= 1 **to** n **do** read (A[i]);

For i:= 1 **to** n **do** read (B[i]);

For i:= 1 **to** n **do**

Begin

If i mod 2 <>0 **then** minim:=Min(A,B,i);

Writeln('de pe pozitiile impare' minim=' , min');

End.

2.1.2. Media Aritmetica a elementelor unui vector, prin funcție.

Type Vector=array [1..100] of integer;

Var n,i: integer;

 a: vector;

 med: real;

Function Media(a1:vector) : real;

Var s,i,n1:integer;

Begin

s:=0;

For i:=1 to n1 **do**

s:=s+a1[i];

Media:=s/n1;

End;

Begin

Readln(n);

For i=1 to n **do** readln (a[i]);

med:=Media(a,n);

Writeln('Media este : ' ,med');

End.

2.1.3. Suma elementelor pare de pe pozitiile impare.

Type Vector=array [1..100] of integer;

Var n,i: integer;

```

        T: vector;
        sum: integer;
Function Suma(V:vector; m:integer ) : integer;
        Var j,s: integer;

Begin
S:=0;
For i:=1 to m do
If (V[j] mod 2=0) and (j mod 2 <>0) then
S:=S+V[j];
Suma:=S;
End;
Begin
Write ('n='); read (n);
For i:= 1 to n do read (T[i]);
Sum:=Suma(T,n);
Writeln('rezultat=' ,sum');
End.

```

2.1.4. Inlocuirea lui DA cu Nu.

```

Program P4;
Var s:string;
    i: integer;
begin
write ('Introdu sirul de caractere'); Readln(s);
For i:=1 to length(s)-2 do

```

If copy (s,i,3)='DA' **then begin**

Delete(s, i, 3);

Insert('NU' ,s, i);

End;

Write('Sirul este: ' ,s');

End.

2.1.5. Stergerea elementelor dintr-un string

Program P5;

Var s: string;

z: char;

i: integer;

begin

write (' Introdu sirul de caractere');

write (' Introdu elementul ce va fi sters'); readln(z);

i:=1;

while i<=length(s) **do**

If s[i]=z **then** delete (s, i, 1) **else** i:=i+1;

Write (' Sirul final ' , s');

End.

2.2. Exemple Recursive

2.2.1. Fibonacci

Program fibonacci;

var n:word;


```

Function fib(n:word):word;
begin
  if n>1 then fib:=fib(n-1)+fib(n-2)
    else fib:=1;
end;
begin
repeat
  write('Locul: '); readln(n);
until n<24;
  writeln ('Numarul de pe locul ',n,' este ',fib(n));
  readln
end.

```

2.2.2. Suma unui sir

```

Var n:integer;
Function P(b:integer): longint;
Begin
If b=0 then P:=0
    else P:= b + P(b-1);
end;
begin;
  readln(n);
  writeln(P(n));

```

end.

2.2.3. Arbore binar

Program Arbore;

type Arbore=^Nod;

Nod=record

Info : string;

Stg, Dr : Arbore;

end;

var T : Arbore;

function Arb : Arbore;

var R : Arbore;

s : string;

begin

readln(s);

if s="" **then** Arb:=nil

else **begin**

new(R); R^.Info:=s;

write('Dați descendentul stîng');

writeln(' al nodului ', s, ':');

R^.Stg:=Arb;

write('Dați descendentul drept');

writeln(' al nodului ', s, ':');

R^.Dr:=Arb; Arb:=R;

end;

end;

Procedure AfisArb(T : Arbore; nivel : integer);

 var i : integer;

begin

if T<>nil **then begin**

 AfisArb(T^.Stg, nivel+1);

for i:=1 **to** nivel **do** write(' ');

 writeln(T^.Info);

 AfisArb(T^.Dr, nivel+1);

end;

end;

begin

 writeln('Dați rădăcina:');

 T:=Arb;

 AfisArb(T, 0);

 readln;

end.

2.2.4 Produsul elementelor.

Type natural=0...maxint;

Var n :natural;

 g, f:real;

Function Fact(nr:natural):real;

Var p:real;

Begin

If n=0 **then** p:=1

Else p:=Fact(nr-1)*nr;

```
        Fact:=P;  
End;  
Begin  
    Readln(n);  
    Writeln(Fact(n));  
End.
```

2.2.5. Reversarea unui string

```
Program Invers;  
Var s: string;  
    I: integer;  
Function sirul (j: integer):string;  
Begin  
    If j=0 then sirul:= ' ' else sir:=s[ j ] = sir( j-1);  
End;  
Begin  
    Writeln(' Introdu sirul'); readln(s);  
    I:=length(S);  
    Writeln(' sirul final este:', sir (i));  
End.
```

3.Diferența dintre iterativitate și recursivitate.

Programele recursive pot fi scrise și nerecursiv, adică sub formă iterativă. Programele recursive nu realizează, de obicei, economie de memorie și nici nu sunt mai rapide în execuție decât variantele lor iterative. Exprimarea recursivă este un mare consumator de timp. Aceasta permite, totuși, o descriere mai compactă și mai clară a subprogramelor, cum se vede în problemele rezolvate mai sus.

Fiind dată o anumită problemă, alegerea între a o rezolva iterativ sau recursiv depinde de cât timp necesită execuția fiecărei variante și de *pretenția de claritate* a textului programului, varianta recursivă producând un text mult mai clar, mai ușor de urmărit. [2]

Studiul comparativ al iterativității și recursivității este reprezentat în tabel

nr. crt	Caracteristici	Iterativitate	Recursivitate
1	Necesarul de memorie	mic	Mare
2	Timpul de execuție	Acelaș	
3	Structura programului	Complicată	Simplă
4	Volumul de muncă	mare	Mic
5	Testarea și depănarea	simplă	complicată

Bibliografie

1. Anatol Gremalshi, Manual de Informatică cl. XI-a, 2014
2. Daniela Oprescu, Manual de Informatică cl. XI-a, 2006
3. https://prezi.com/qfmfcl_7jdpg/recursivitate-si-iterativitate/
4. <http://info.tm.edu.ro:8080/~junea/cls%2010/recursivitate/recursivitate.pdf>
5. <https://www.pbinfo.ro/articole/3873/recursivitate>