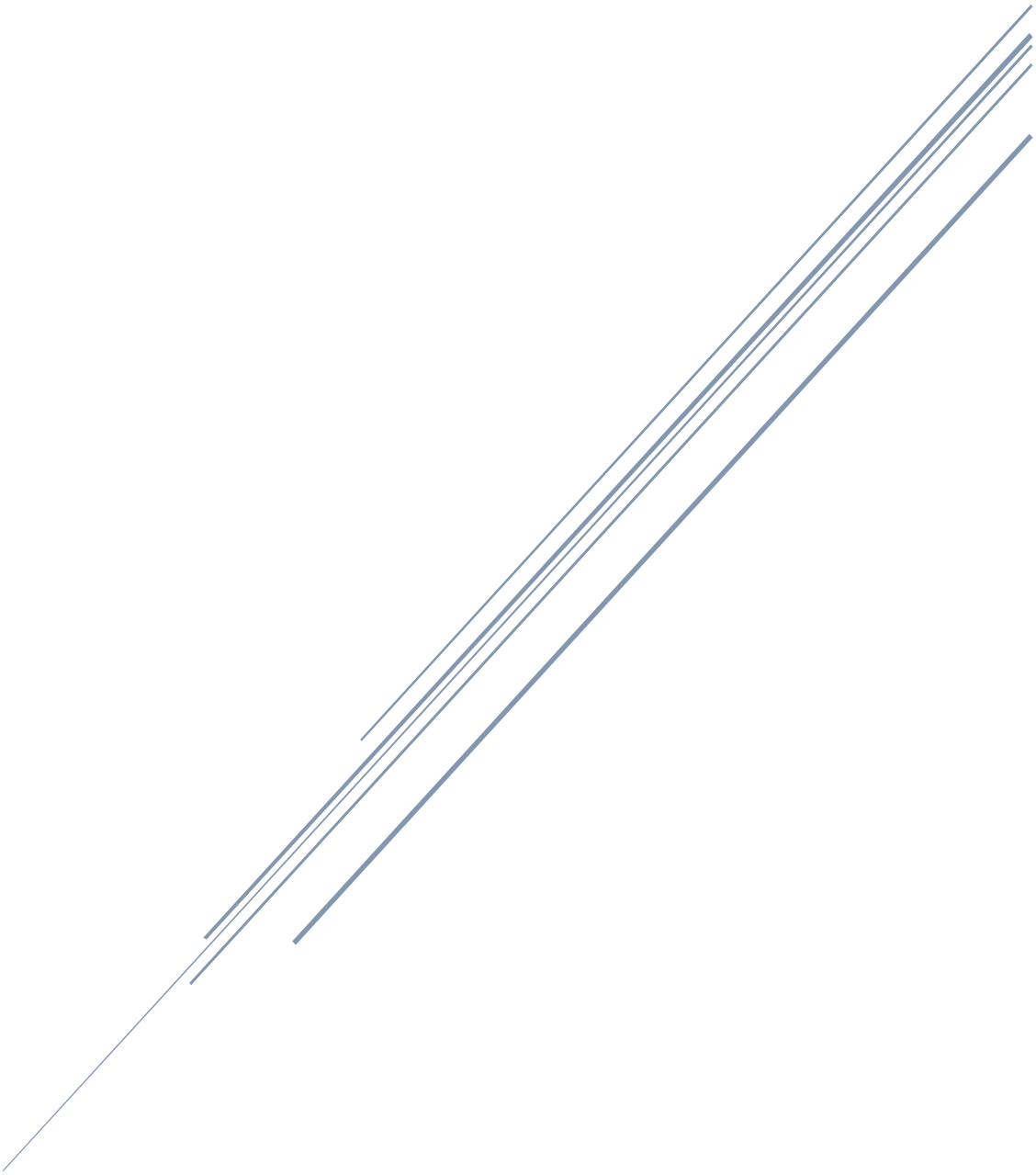
ITERATIVITATE ȘI RECURSIVITATE



Tataru Vlada, clasa a XI-a “D”
30.03.2020

Cuprins

1. Descrierea metodei

1.1. Iteravitatea

http://www.authorstream.com/Presentation/aSGuest41792-360322-iterativitate-sau-recursivitate-rodika-guzun-science-technology-ppt-powerpoint/
https://www.slideserve.com/sveta/algoritmi-i-scheme-logice
https://www.slideshare.net/Vytamin/iterativitate-sau-recursivitate

Iterativitatea este procesul prin care rezultatul este obţinut ca urmare a execuţiei repetate a unui set de operaţii, de fiecare dată cu alte valori de intrare. Numărul de iteraţii poate fi necunoscut sau cunoscut, dar determinabil pe parcursul execuţiei. Metoda de repetivitate este cunoscută sub numele de ciclu (loop) şi poate fi realizată prin utilizarea următoarelor structuri repetitive: ciclul cu test iniţial, ciclul cu test final, ciclul cu număr finit de paşi. Indiferent ce fel de structură iterativă se foloseşte este necesar ca numărul de iteraţii să fie finit.[1]

Iteratia este execuţia repetată a unei porţiuni de program pînă la îndeplinirea unei condiţii(while,for etc.) Dupa cum sa vazut,orice algoritm recursiv poate fi transcris intr-un algoritm iterativ si invers.Alegerea tehnicii de programare-iterativitatea sau recursivitatea- tine de competenta programatorului.[1]

Cu ajutorul iteravitații vei putea calcula produsul vectorial, pătratele elementelor unui șir și vei mai putea crea vectori. [2]

Caracteristici, avantaje și dezavantaje:

1. Necesar de memorie mic;
2. Structura programului este complicată;
3. Volum de muncă mare;
4. Testare si depănare simplă.[3]

1.2. Recursivitatea

https://prezi.com/qfmfcl_7jdpg/recursivitate-si-iterativitate/
https://www.slideserve.com/sveta/algoritmi-i-scheme-logice
https://ru.scribd.com/document/337119802/Iterativitatea

Recursivitatea este procesul iterativ prin care valoarea unei variabile se determină pe baza uneia sau a mai multora dintre propriile ei valori anterioare.Structurile recursive reprezintă o alternativă de realizare a proceselor repetitivefără a utiliza cicluri.

Mecanismul recursivitatii consta in posibilitatea ca unsubpro□ram sa se autoapeleze.
Exista două tipuri de recursivitate:

1) recursivitate directă - când un subprogram se autoapelează în corpul său;
2) recursivitate indirectă - când avem două subprograme (x si y), iar x face apel la y şi invers;

Se folosesc algoritmi recursivi atunci c!nd calculele aferente sunt descrise în formă recursivă.

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.

Pentru a permite apelarea recursivă a subprogramelor, limbajul Pascal dispune de mecanisme speciale de suspendare a execuţiei programului apelant, de salvare a informaţiei necesare şi de reactivare a programului suspendat . [5]

Cu ajutorul **recursvității** poți calcula suma elementelor unui șir, produsul elementelor unui șir, produsul scalar, maxim/minim dintr-un șir și cel mai mare/mic din două numere.[3]

Apelul recursiv al unei proceduri (functii) face ca pentru toti parametrii-valoare sa se creeze copii locale apelului curent (in stiva) , acestea fiind referite si asupra lor facindu-se modificarile in timpul executiei curente a procedurii (functiei). Cind executia procedurii (functiei) se termina, copiile sint extrase din stiva, astfel incit modificarile operate asupra parametrilor-valoare nu afecteaza parametrii efectivi de apel, corespunzatori.[4]

De asemenea pentru toate variabilele locale se rezerva spatiu la fiecare apel recursiv.[4]

Caracteristici, avantaje şi dezavantaje:

1. Necesar de memorie mare;
2. Structura programului simplă;
3. Volum de muncă mic;
4. Testare și depănare complicată.[3]

2. Probleme rezolvate

2.2. Probleme iterative

https://manuale.edu.ro/manuale/Clasa%20a%20XI-a/Informatica/Niculescu2/A330.pdf?fbclid=IwAR1i4WgLK-6MJ82ycy09DBqRsFNDN5cK92MfwhNqVmuaFuU2y-arQxw61WM

1. **Suma cifrelor unui număr**

```
function suma_cifre(n:word):byte;
var sum:byte;
begin sum:=0;
while n<>0 do begin
            sum:=sum + n mod 10;
            n:=n div 10;
end;
suma_cifre:=sum      end;
```

2. **a^b prin înmulțire repetată**

```
function putere(a,b:word):word;
var p:word;
begin p:=1;
while b<>0 do begin
              p:=p * a;
              b:=b – 1;
              end;
 putere:=p        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. **CMMDC**

```
function cmmdc(a,b:word):word;
var rest:word;
begin rest:=a;
while rest <> 0 do
begin rest:=a mod b;  a:=b; b:=rest;
end;
```

```
cmmdc:=a; end;
```

5. Factorial

```
var n:byte;
function fact(n:byte):word;
var i:byte;
calcul:word;
begin calcul:=1;
for i:=2 to n do calcul:=calcul * i;
 fact:=calcul;
end;
begin
readln(n);
writeln('Fact=',fact(n)); end.
```

2.1. Probleme recursive

https://manuale.edu.ro/manuale/Clasa%20a%20XI-a/Informatica/Niculescu2/A330.pdf?fbclid=IwAR1i4WgLK-6MJ82ycy09DBqRsFNDN5cK92MfwhNqVmuaFuU2y-arQxw61WM

1. **Suma cifrelor unui număr**

```
function suma_cifre(n:word):byte;
begin
if n=0 then suma_cifre:=0
        else suma_cifre:=n mod 10 + suma_cifre (n div 10);
end;
```

2. **a^b prin înmulþire repetată**

```
function putere(a,b:word):word;
begin
if b=0 then putere:=1
        else putere:=a*putere(a,b-1);
end;
```

3. **a*b prin adunare repetată**

```
function produs(a,b:word):word;
begin
if b=0 then produs:=0
       else produs:=a+produs(a,b-1);
end;
```

4. **Suma unui șir**

```
var n:integer;
Function F (N:Integer):Longint;
Begin
If N=0 Then F:=0
Else F:=F(N-1)+N
End;
begin
readln(n);
writeln(F(n));
end.
```

5. Aflarea elemntelor max și min

```
type vector=array[1..100]of integer;
var n,i:byte; a:vector;
function Max (i,j:integer;var a:vector):integer;
var max1,max2:integer;
begin
if (i=j) then Max:=a[i]
else if (i=j-1) then
begin
if a[i]<a[j] then Max:=a[j]
else Max:=a[i];
end
else begin
max1:=Max(i,(i+j)div 2,a);
max2:=Max((i+j)div 2,j,a);
if max1>max2 then Max:=max1
else Max:=max2;
end; end;
```

3. Concluzii:

Iterativitatea și recursivitatea reprezintă o tehnică de programare de o importanță deosebită. Ele permit o experimare extrem de concisă și clară a algoritmilor de rezolvare a unor probleme complexe. Orice algoritm recursiv poate fi transcris într-un algoritm iterativ și invers,timpul de executie fiind același, ceea ce înseamnă că alegerea tehnicii de programare ține de competența fiecarui programator în parte.

Primele 3 probleme au fost făcute în paralel, în modul recursiv și în cel iterativ. Obervăm totuși că programul în recursiv este într-o forma mai compactă și mai rapid de realizat. Varianta recursivă produce un text mult mai clar și mai ușor de urmărit.

4. Bibliografie

http://www.authorstream.com/Presentation/aSGuest41792-360322-iterativitate-sau-recursivitate-rodika-guzun-science-technology-ppt-powerpoint/ [1]

https://www.slideserve.com/sveta/algoritmi-i-scheme-logice [2]

https://www.slideshare.net/Vytamin/iterativitate-sau-recursivitate [3]

https://prezi.com/qfmfcl_7jdpg/recursivitate-si-iterativitate/ [4]

https://ru.scribd.com/document/337119802/Iterativitatea [5]

https://manuale.edu.ro/manuale/Clasa%20a%20XI-a/Informatica/Niculescu2/A330.pdf?fbclid=IwAR1i4WgLK-6MJ82ycy09DBqRsFNDN5cK92MfwhNqVmuaFuU2y-arQxw61WM [6]