ITERATIVITATEA SAU RECURSIVITATEA

Bulhac Bogdan-Grigorie

1. Aspecte tepretice - 1 -
1.1. Iterativitatea - 1 -
1.2. Recursivitatea - 1 -
Realizarea autoapelului - 1 -
Observaţii - 2 -
Cum gândim un algoritm recursiv? - 2 -
2. Exemple - 2 -
2.1. Exemple Iterative - 2 -
2.1.1. Stergerea elementelor dintr-un string - 2 -
2.1.2. Sita lui Eratostene - 2 -
2.1.3. Ordonarea in ordine crescatoare a numerelor unui tablou - 3 -
2.1.4. Inlocuirea lui sau cu ori - 3 -
2.1.5. Parcurgerea arborelui binar - 4 -
2.2. Exemple Recursive - 5 -
2.2.1. Suma unui sir - 5 -
2.2.2. Reversarea unui string - 5 -
2.2.3. aflarea elementelor max si min - 6 -
2.2.4. Parcurgerea arborilor - 6 -
2.2.5. Calcularea unui polinom intr-un punct xO real - 7 -
3.Diferențe între algoritmi iterativi și recursivi - 8 -
Bibiliografie - 11 -

1. Aspecte tepretice

1.1. Iterativitatea

Definitia: [3]

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.

[5] Toţi algoritmii iterativi pot fi înlocuiţi prin algoritmi recursivi. Avantajul celor recursivi este dat de scăderea dimensiunilor programelor şi de creşterea lizibilităţii. Avantajul celor iterativi este viteza mărită de execuţie prin gestionarea locală a parametrilor de ciclare (eliminîndu-se astfel toate instrucţiunile push şi pop pentru gestionarea stivei).

1.2. Recursivitatea

Definitia: [3]

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).

[6] Recursivitatea reprezintă o tehnică de programare de o importanţă deosebită. Ea permite o exprimare extrem de concisă şi clară a algoritmilor de rezolvare a unor probleme complexe.
Un subprogram este recursiv dacă se apelează pe el însuşi (se autoapelează).

Realizarea autoapelului

- În cazul funcţiilor de tip *void*, autoapelul se realizează prin apelul funcţiei respective, din interiorul ei. Apelul se face la fel ca în cazul în care funcţia este apelată din exterior.
- În cazul funcţiilor care nu sunt de tip *void*, autoapelul se realizează prin instrucţiunea *return*. Ea este de forma *return expresie* dar în expresia respectivă trebuie să intre şi funcţia care se autoapelează.

Observaţii

- Pentru orice algoritm recursiv există unul iterativ care rezolvă aceeaşi problemă.
- Nu întotdeauna alegerea unui algoritm recursiv reprezintă un avantaj.
- Recursivitatea presupune mai multă memorie în comparaţie cu iterativitatea.

Cum gândim un algoritm recursiv?

- Ce se întâmplă la un nivel se întâmplă la orice nivel.
- Subprogramul care se autoapelează trebuie să conţină instrucţiunile corespunzătoare unui nivel.
- Starea tratată de subprogram se găseşte pe un anumit nivel al stivei.

2. Exemple

2.1. Exemple Iterative

2.1.1. Stergerea elementelor dintr-un string

```
Program sterge_x;
var s: string; x: char;
i:integer;
begin
write ('Introdu sirul de caractere '); readln(s);
write ('Introdu ce va fi sters '); readln(x);
i:=1;
while i<= length(s) do
if s[i]=x then delete(s, i, 1) else i:=i+1;
write ('Sirul modificat ', s);
end.
```

2.1.2. Sita lui Eratostene

```
type m = set of 1..100;
var s, np : m;
procedure Prim( sita, nrp : m);
var k, i, n : integer;
begin
readln(k);
sita:=[2..k];
nrp:=[];
i:=2;
repeat
while not (i in sita) do i:=succ(i);
nrp:=nrp+[i];
```

```
write(i,' ');
n:=i;
while n<=k do
begin
sita:=sita-[n];
n:=n+i;
end;
until sita=[];
writeln;
end;
begin
Prim(s,np);
end.
```

2.1.3. Ordonarea in ordine crescatoare a numerelor unui tablou

```
type tbl=array[1..100] of real;
var o : tbl;
    j, m : integer;
procedure Tabl( a : tbl; n : integer);
var i, pr : integer; h : real;
begin
repeat
pr:=0;
for i:=1 to n-1 do
begin
if a[i]>a[i+1] then begin
h:=a[i]; a[i]:=a[i+1]; a[i+1]:=h; inc(pr);
end; end;
until pr=0;
writeln;
for i:=1 to n-1 do
write (a[i],','); write(a[n]);
end;
begin
write('n=...'); read(m);
for j:=1 to m do read (o[j]);
for j:=1 to m-1 do
write (o[j],','); write(o[m]);
Tabl(o, m);
end.
```

2.1.4. Inlocuirea lui sau cu ori

```
Program inlocuieste_sau_cu_ori;
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)='sau' then begin
delete(s,i,3);
```

```
insert('ori',s, i);
end;
write ('Sirul obtinut este: ',s);
end.
```

2.1.5. Parcurgerea arborelui binar

[1]

```
Program P131;
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('Daţi descendentul stîng');
writeln(' al nodului ', s, ':');
R^.Stg:=Arb;
write('Daţi 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;
procedure Preordine(T : Arbore);
begin
if T<>nil then begin
writeln(T^.Info);
Preordine(T^.Stg);
Preordine(T^.Dr)
end;
end;
procedure Inordine(T : Arbore);
begin
if T<>nil then begin
Inordine(T^.Stg);
writeln(T^.Info);
Inordine(T^.Dr)
```

```
end;
end;
procedure Postordine(T : Arbore);
begin if T<>nil then begin
Postordine(T^.Stg);
Postordine(T^.Dr);
writeln(T^.Info)
end;
end;
begin
writeln('Daţi rădăcina:');
T:=Arb;
AfisArb(T, 0);
readln;
writeln('Parcurgere în preordine:');
Preordine(T);
readln;
writeln('Parcurgere în inordine:');
Inordine(T);
readln;
writeln('Parcurgere în postordine:');
Postordine(T);
readln;
end.
```

2.2. Exemple Recursive

2.2.1. Suma unui sir

```
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.
```

2.2.2. Reversarea unui string

```
Program String_invers;
Var s: string;
    j: integer;

Function sir(i: integer):string;
Begin
```

```
if i=0 then sir:='' else sir:=s[i]+sir(i-1);
End;

Begin
Writeln('Introdu sirul'); readln(s);
j:=length(s);
Writeln('sirul inversat este: ', sir(j));
End.
```

2.2.3. aflarea elementelor max si 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;

begin
write('n='); readln(n);
writeln;
for i:=1 to n do read(a[i]);
writeln;
write('elementul maxim este...', Max(1,n,a));
end.
```

2.2.4. Parcurgerea arborilor

[2]

```
program parcurg_arbore;
 var s,d:array[1..10]of byte; {globale}
procedure RSD(k:byte);
 begin write(k,' ');
 if s[k]<>0 then RSD(s[k]);
if d[k]<>0 then RSD(d[k]);
 end;
procedure SRD(k:byte);
 begin if s[k]<>0 then SRD(s[k]);
 write(k,' ');
if d[k]<>0 then SRD(d[k]);
 end;
procedure SDR(k:byte);
```

```
 begin if s[k]<>0 then SDR(s[k]);
 if d[k]<>0 then SDR(d[k]);
write(k,' ');
end;
var n,i,rad:byte;

begin {p.p.}
write('Dati numarul de noduri n: ');
readln(n);
 write('Introduceti nodul radacina: ');
 readln(rad);
for i:=1 to n do
 begin
write(' Pentru nodul ',i);
writeln(' introduceti');
writeln('descendent stang: ');
 readln(s[i]);
 writeln('descendent drept: ');
 readln(d[i]);
 end;
 writeln;
writeln(' * Parcurgere in preordine (RSD) **');
 RSD(rad);
writeln(' * Parcurgere in inordine (SRD) **');
SRD(rad);
writeln(' * Parcurgere in postordine (SDR) **');
SDR(rad);
readln;
end.
```

2.2.5. Calcularea unui polinom intr-un punct xO real

[2]

```
var a:array[0..10]of real;
n,i:integer;
x0:real;
 function eval(n:integer):real;
 begin
if n<0 then eval:=0
else eval:=a[n]+x0*eval(n-1);
 end;
begin {p.p.}
write('Gradul polinomului: ');
readln(n);
write('coeficientii:');
for i:=0 to n do
begin
write('a[',n-i,']=');
readln(a[i])
end;
write('Valoarea de evaluare:');
readln(x0);
writeln('Valoarea polinomului: ' ,eval(n):8:3);
end.
```

3.Diferențe între algoritmi iterativi și recursivi

[4]

Atât iterarea, cât şi recursul se bazează pe o structură de control: Iteraţia foloseşte o structură de repetiţie; recursiunea foloseşte o structură de selecţie. Un algoritm iterativ va folosi instrucţiuni de buclă, cum ar fi pentru buclă, în timp ce buclă sau buclă pentru a repeta aceiaşi paşi, în timp ce un algoritm recursiv, un modul (funcţia) se recheamă din nou şi din nou până când condiţia de bază (condiţia de oprire) este îndeplinită.

Un algoritm Iterativ va fi mai rapid decât algoritmul recursiv din cauza cheltuielilor generale cum ar fi funcţiile de apel şi stivele de înregistrare în mod repetat. De multe ori, algoritmii recursivi nu sunt eficienţi deoarece necesită mai mult spaţiu şi timp.

Algoritmii recursivi sunt utilizaţi în cea mai mare parte pentru a rezolva probleme complicate atunci când aplicarea lor este uşoară şi eficientă. De exemplu, algoritmul Tower of Hannoi este uşor de recurs, în timp ce iteraţiile sunt utilizate pe scară largă, eficiente şi populare.

Algoritmi recursivi vs. iterativi:

- **Abordare**: În abordarea recursivă, funcţia se solicită până când condiţia este îndeplinită, în timp ce în abordarea iterativă se repetă o funcţie până când condiţia nu reuşeşte.
- Utilizarea **programelor de construcţie**: Algoritmul recursiv utilizează o structură de ramificaţie, în timp ce algoritmul iterativ utilizează o construcţie looping.
- **Eficienţa timpului şi a spaţiului**: soluţiile recursive sunt adesea mai puţin eficiente din punct de vedere al timpului şi spaţiului, comparativ cu soluţiile iterative.
- **Test de terminare**: Iteraţia se termină atunci când condiţia continuă a buclăului eşuează; recursiunea se termină când se recunoaşte un caz de bază.
- **Invitaţie infinită**: Se produce o buclă infinită cu iteraţie dacă testul de continuare a buclării nu devine fals; se produce recurenţa infinită dacă etapa de recurs nu reduce problema într-o manieră care converge în cazul de bază.

O problemă practică: tipul de ziua de naştere taie tortul de ziua şi trebuie să se asigure că toată lumea din cameră primeşte o felie.

Soluţia 1 – Iterativă: tipul care îşi serbează ziua de naştere taie tortul şi se asugră ca toatş lumea primeşte câte o felie.

Serviţifelia (în timp ce există oameni pentru a servi sau felii rămase)

Soluţia 2 – Recursivă: Luaţi o felie de tort de pe tavă şi treceţi tava următoarei persoane care ia o felie din tavă şi trece tava persoanei următoare, care ia o felie din tavă şi trece tava la persoana următoare …

Reţineţi că aceeaşi funcţie este efectuată de fiecare data.

Ia felia (tava) Dacă sunt oameni care servesc sau felii rămase Ia felia si and şi invite să servească (tava) altfel revin-o.

[2]

PROCES	RECURSIV	ITERATIV

FACTORIAL	`var n:byte;`
	`function fact(n:byte):word;`
	`begin`
	`  if n=0 then fact:=1`
	`  else fact:=n*fact(n-1);`
	`end;`
	`begin {p.p.}`
	`readln(n);`
	`writeln('Fact=,fact(n));`
	`end.`
		`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.`

CMMDC	`function cmmdc(a,b:word):word;`
	`begin`
	`if b<>0 then cmmdc:=cmmdc(b,a mod b)`
	`else cmmdc:=a;`
	`end;`
		`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;`

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;`
		`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;`

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

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;`
		`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;`

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.`
		`var n:word;`
		`function fib(n:word);word;`
		`var i:byte;prec1,prec2,curent:word;`
		`begin`
		`prec1:=1; curent:=1;`
		`i:=3;`
		`while i<=n do begin`
		`prec2:=prec1;`
		`prec1:=curent;`
		`curent:=prec1+prec2;`
		`i:=i+1`
		`end;`
		`fib:=curent`
		`end;`
		`begin`
		`readln(n);`
		`writeln('Pe locul ',n,' se afla ',fib(n))`
		`end.`

Bibiliografie

1. Anatol Gremalschi file:///C:/Users/User/Downloads/XI_Informatica%20(in%20limba%20romana).pdf
2. Daniela Oprescu https://manuale.edu.ro/manuale/Clasa%20a%20XI-a/Informatica/Niculescu2/A330.pdf
3. Proiect elaborate de Lisnic Valeria https://prezi.com/qfmfcl_7jdpg/recursivitate-si-iterativitate/
4. https://www.codeit-project.eu/ro/differences-between-iterative-and-recursive-algorithms/
5. http://staff.fmi.uvt.ro/~victoria.iordan/Programare_MI/Programming%20-%20Culegere%20probleme%20de%20informatica.pdf
6. https://innf.weebly.com/recursivitate.html