ITERATIVITATE SAU RECUSIVITATE?

felicia.burlacu1@gmail.com
11 D | FELICIA BURLACU

ITERATIVITATE SAU RECURSIVITATE

1. ITERATIVITATE. DEFINITII SI CONCEPTE DE BAZA.

Iterativitatea
este procesul prin care rezultatul este obţinut ca urmare aexecuţiei 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 parcursulexecuţiei. Metoda de repetivitate este cunoscută sub numele de ciclu (loop) şi poatefi realizată prin utilizarea următoarelor structuri repetitive: ciclul cu test iniţial,ciclul cu test final, ciclul cu număr finit de paşi. Indiferent ce fel de structurăiterativă se foloseşte este necesar ca numărul de iteraţii să fie finit.[1]

1.1 Exemplu de problema utilizand algoritmul iterativ:

//subprogramul va numara cate cifre are un numar natural n

```
function nrdigit(n:Natural):integer;
var k:integer;
begin
  repeat
  begin n:=n div 10 ;
  inc(k);
  end;
  until n = 0;
  nrdigit:=k;
end;
```

2. RECURSIVITATE. DEFINITII SI CONCEPTE DE BAZA.

Recursivitatea
este procesul iterativ prin care valoarea unei variabile sedetermină 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 unsubprogram sa se autoapeleze

Recursivitatea este frecvent folosită in prelucrarea structurilor de date definiterecursiv . Un subprogram recursiv trebuie scris astfel incat să respecte regulile :

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

b)Subprogramul recursiv se va autoapela intr-un mod in care se tinde spre ajungerea in situaţia de execuţie fără autoapel. [2]

2.1 Exemplu de program utilizand algoritmul recursiv

//subprogramul va numara cate cifre are un numar natural n

```
1. type Natural = 0..MaxLongInt;
2. var n:Natural;
3. function NrCifre(n:Natural):Natural;
4. begin
5. if n in [0..9]
6. then NrCifre:=1
7.  else NrCifre:=1 + NrCifre(n div 10);
8. end;
```

3. ANALIZA COMPARATIVA A 2 PROBLEME REZOLVATE.

3.1 Programul elborat va verifica daca numarul n introdus de la tastatura este palidrom sau nu.

METODA ITERATIVA

```
program polidrom;
var n,i,m: integer;
    x:boolean;

function nrdigit(m:integer):integer;
var k:integer;
begin
  repeat begin
    m:= m div 10 ;
    inc(k); end;
  until m=0 ;
  nrdigit:=k;
end;

function polindrom(m:integer):boolean;
var i:integer;
s:string;
begin
  str(m,s);
  for i := 1 to nrdigit(m) div 2 do begin
    if s[i]=s[nrdigit(m)-i+1] then polindrom := true else polindrom := false;
  end;
end;

begin
  writeln('dati n'); readln(n);
  x:=polindrom(n);
  writeln(x);
      end.
```

METODA RECURSIVA

```
program Palindrom;

type Natural = 0..MaxInt;
var n,nrcif:Natural;

function NrCifre(n:Natural):Natural;
begin
 if n in[0..9] then NrCifre:=1
```

```
 else NrCifre:=1+NrCifre(n div 10);
end;


function Oglinda (n:Natural): Natural;
var aux:Natural;
begin
 if n in [0..9]
 then Oglinda:= n
 else
 begin
 aux :=NrCifre(n div 10);
 Oglinda:=trunc((n mod 10)*exp(aux*ln(10))) + Oglinda(n div 10);
 end;
end;

begin
 write ('dati n='); readln(n);
 nrcif :=NrCifre(n);
 writeln('oglinditul=',Oglinda(n));
 if n=Oglinda(n)
 then writeln (n,' este palindrom ')
 else writeln (n,' nu este palindrom');

end.
```

3.2 programul elaborate va afisa pozitia unei cifre in cadrul unui numar

```
1. program Pozit;
2. type Natural = 0..MaxInt;
3. var n,Poz:Natural;
4.     x,rsp:integer;
5.
6. function nrdigit(n:Natural):integer;
7. var k:integer;
8. begin
9.    repeat
10.         begin n:=n div 10 ;
11.         inc(k);
12.         end;
13.         until n = 0;
14.         nrdigit:=k;
15.       end;
16.
17.       function nrpoz(n:Natural; x:Natural):integer;
18.       var l,i:integer;
19.           aux,aux1:string;
20.       begin
21.         str(n,aux);
22.         str(x,aux1);
23.         l:=nrdigit(n);
24.         for i:= 1 to l do if aux[i] = aux1 then nrpoz:=i;
25.       end;
26.
27.       begin
28.         writeln('n='); readln(n);
29.         writeln('x='); readln(x);
30.         rsp:=nrpoz(n,x); writeln(rsp);
31. end.
```

METODA RECURSIVA

```
program Pozit;
type Natural = 0..MaxInt;
var n,Poz:Natural;

function Pozitia (n,Poz:Natural): byte;
begin
 if Poz=1
 then Pozitia:=n mod 10
 else
 begin
 Poz:=Poz-1;
 Pozitia:=Pozitia(n div 10,Poz);
 end;
end;

begin
 write ('dati n=');
 readln (n);
 write ('a cata cifra =');
 readln (Poz);
 writeln (”cifra este:”,Pozitia(n,Poz));
end.
```

4. EXEMPLE DE PROBLEME REZOLVATE(3)

4.1 ITERATIVITATE

1. Programul elaborate va verifica dacă trei numere a,b,c reale pot reprezenta laturile unui triunghi. Dacă da, să se caculeze perimetrul şi aria sa.

```
 Program Laturile_Unui_Triunghi;
Var a,b,c,s,p:real;
function laturi_ok:boolean;
begin
 laturi_ok:= (a>0) and (b>0) and (c>0) and (a+b>c) and (a+c>b) and (b+c>a) ;
end;
begin
write('introduceti laturile');readln(a,b,c);
if laturi_ok then
begin
 p:=(a+b+c)/2;
 s:=sqrt(p*(p-a)*(p-b)*(p-c));
 writeln('Aria=',s:5:2);
 writeln('Perimetrul=',2*p:5:2);
end
else writeln('Nu formeaza triunghi');
readln;
end.
```

2.programul elaborate va rezolva o functie de gradul II

```
Var a,b,c,delta:real;
BEGIN
Write('Introd. a,b,c:');Readln(a,b,c);
delta:=b*b-4*a*c;
If delta>=0 then
 Begin
 Writeln('x1=',(-b-sqrt(delta))/(2*a):6:2);
 Writeln('x2=',(-b+sqrt(delta))/(2*a):6:2);
22
 End
 else Begin
 Writeln('z1=(',-b/(2*a):6:2, ',' , -sqrt(-delta))/(2*a):6:2, ')');
 Writeln('z2=(', -b/(2*a):6:2, ',' , sqrt(-delta))/(2*a):6:2, ')');
 End
Readln;
END.
```

3.programul elaborat determina al n-lea numar din sirul Fibonacci

```
{
Determină al n
-lea numar fibonacci.
In:
- n: numarul fibonacci dorit.
Out:
- al
n
-lea numar fibonacci.
}
function fibo1(n:integer):integer;
var prev,crt,f,i:integer;
begin
 prev:=0; crt:=1;
 for i:=1 to n
-1 do
 begin
 f:=crt+prev;
 prev:=crt;
 crt:=f;
 end;
 fibo1:=crt;
end; {
Determină al n
-lea numar fibonacci.
In:
- n: numarul fibonacci dorit.
Out:
- al n
-lea numar fibonacci.}
function fibo2(n:integer):integer;
var prev,crt,i:integer;
begin
 prev:=0;crt:=1;
 for i:=1 to n div 2 do
 begin
 prev:=prev+crt;
 crt:=crt+prev;
 end;
 if n mod 2 = 0
 then fibo2:=prev
 else fibo2:=crt;
end;
```

4.2 RECURSIVITATE

1. programul elaborate citeste caracterele tastate pe o linie, tiparindu-le apoi in ordine inversa.

```
1    program var_recursiva;
2.       procedure prel_car;
3.            var car:char;
4.            begin
5.               read(car);
6.               if not eoln then prel_car;
7.               write(car)
8.            end;
9.       begin
10.                prel_car
11.      end.
```

2. Program elaborat calculeaza 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.
```

3. Programul elaborat parcurge un arbore

```
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);
- 7 -
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);
   end.
```

5. CONCLUZII. ANALIZA COMPARATIVA A ALGORITMILOR.

- Abordare: În abordarea recursivă, funcția se solicită până când condiția este îndeplinită, în timp ce în abordarea iterativă se repetă o funcție până când condiția nu reușește.
- Utilizarea programelor de construcție: Algoritmul recursiv utilizează o structură de ramificație, în timp ce algoritmul iterativ utilizează o construcție looping.
- Eficiența timpului și a spațiului: soluțiile recursive sunt adesea mai puțin eficiente din punct de vedere al timpului și spațiului, comparativ cu soluțiile iterative.
- Test de terminare: Iterația se termină atunci când condiția continuă a buclăului eșuează; recursiunea se termină când se recunoaște un caz de bază.
- Invitație infinită: Se produce o buclă infinită cu iterație dacă testul de continuare a buclării nu devine fals; se produce recurența infinită dacă etapa de recurs nu reduce problema într-o manieră care converge în cazul de bază.