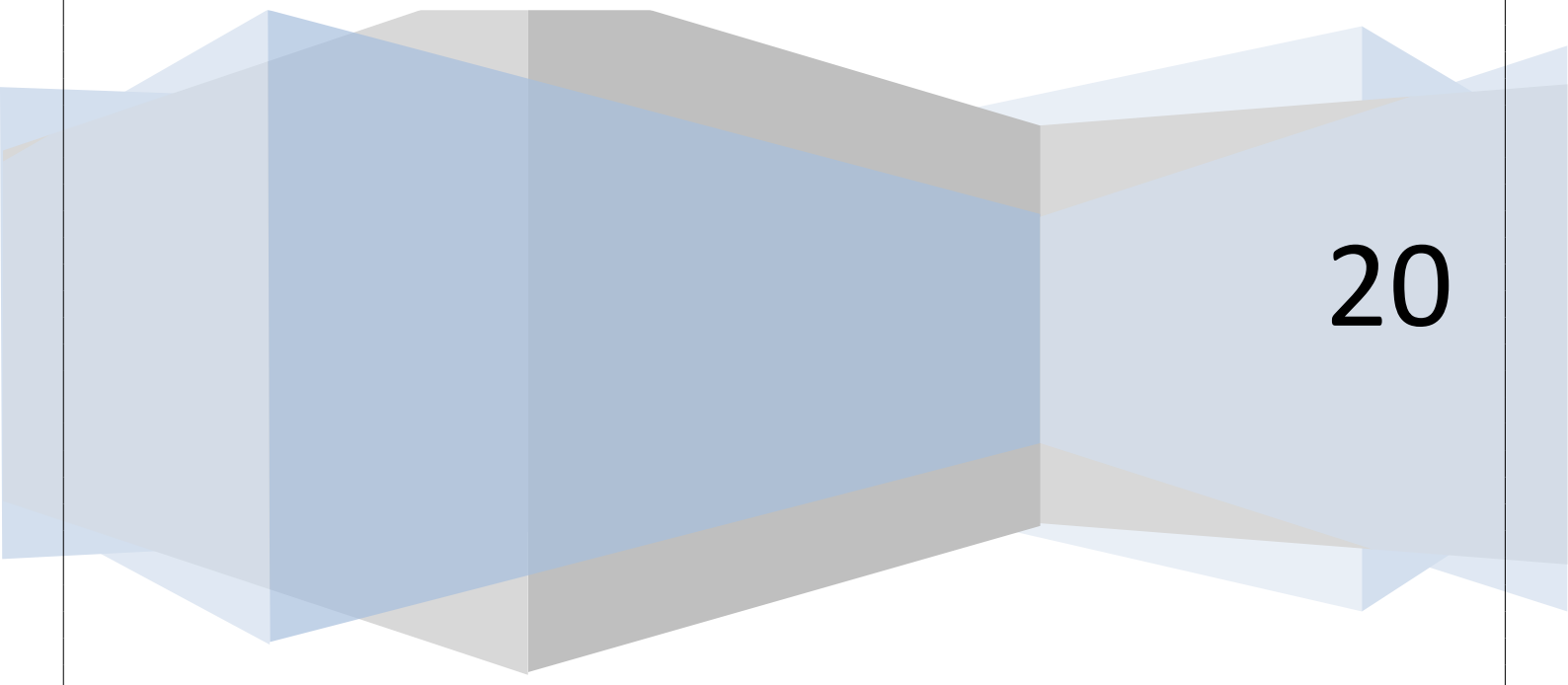
IPLT „Spiru Haret”

Referat

Iterativitate si Recursivitate

Efectuat de: Buzu Alexandru cl. 11”D”

Verificat de: Guțu Maria



20

Cuprins

Iterativitatea

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 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.

- **Iteraţia** este execuţia repetată a unei porţiuni de program până la îndeplinirea unei condiţii(while,for etc.)[1]

Recursivitatea

Recursivitatea este un mecanism general de elaborare a algoritmilor. O funcție se numește recursivă dacă ea se autoapelează, fie *direct* (în definiția ei, se face apel la ea însăși), fie *indirect* (funcția X apelează funcția Y, care apelează funcția X).

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 repetitive fără a utiliza cicluri.

Recursivitatea a aparut din necesități practice date de transcrierea directă a formulelor matematice recursive. Apoi, acest mecanism a fost extins, fiind utilizat în elaborarea multor algoritmi.

Recursivitatea este frecvent folosită în prelucrarea structurilor de date definite recursiv. Un subprogram recursiv trebuie scris astfel încât să respecte regulile :

a) subprogramul trebuie să poată fi executat cel puțin o dată fără a se autoapela
b) subprogramul recursiv se va autoapela într-un mod în care se tinde spre ajungerea în situația de execuție fără autoapel.

Se știe că la apelul fiecarui subprogram, se generează un nou nivel în segmentul de stiva, corespunzător acelui apel. Pe acel nivel se memorează :

1. valorile parametrilor transmiși prin valoare
2. adresele parametrilor transmiși prin referință
3. variabilele locale
4. adresa de revenire din funcție

Analiza comparativă a metodelor

- **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.[3]
- **Structura programului** este mai complicată pentru algoritmul iterativ comparativ cu cel recursiv.
- **Volumul de muncă necesar pentru scrierea programului** este mai mic în cazul recursiei în comparaţie cu iteraţia.
- **Testarea şi depanarea programelor** este mai simplă pentru algoritmul iterativ comparativ cu cel recursiv.[4]

Exemple rezolvate

• Exemple iterative

1. {calcularea lui x la puterea **n** }[4]

```
type Natural=0..MaxInt;
function Putere(x : real; n : Natural) : real;
 var p : real;
      i : integer;
begin
    p:=1;
    for i:=1 to n do p:=p*x;
    Putere:=p;
end;
```

2. {calcularea factorialului numărului **n** }[4]

```
function Factorial(n : integer) : integer;
var p, i : integer;
begin
   p:=1;
   for i:=1 to n do p:=p*i;
   Factorial:=p;
end;
```

3. { Calculatea sumei 1 + 1/2 + 1/3 + ... + 1/**n** }[6]

```
Program P62;
var n, i : 1..MaxInt; s : real;
begin
   write('n='); readln(n);
   s:=0;
   for i:=1 to n do s:=s+1/i;
   writeln('s=', s);
end.
```

4. { Tabelul functiei y=2*x }[6]

```
Program P65;
var x, y, x1, x2, deltaX : real;
begin
    write('x1='); readln(x1);
    write('x2='); readln(x2);
    write('deltaX='); readln(deltaX);
    writeln('x':10, 'y':20);
    writeln; x:=x1;
    while x<=x2 do
       begin
          y:=2*x;
          writeln(x:20, y:20);
          x:=x+deltaX;
       end;
end.
```

5. { Media aritmetica a **n** numere }[6]

```
Program P64;
var x, suma, media : real;
    i, n : integer;
begin
 write('n='); readln(n);
 suma:=0;
 writeln('Dati ', n, ' numere:');
 for i:=1 to n do
   begin
    write('x='); readln(x);
    suma:=suma+x;
   end;
 if n>0 then
          begin
            media:=suma/n;
            writeln('media=', media);
          end
        else writeln('media=*****');
  readln;
end.
```

• Exemple recursive[5]

1. Determinarea numărului de cifre a unui număr natural **n:**

-pentru **n∈ {0,1,2,3,4,5,6,7,8,9}** evident numărul de cifre este **1**;

-pentru **n ≥ 10**, se poate gândi că numărul de cifre a lui **n** este **1**+ numărul de cifre a numărului obţinut din **n**, din care am tăiat ultima cifră (de fiecare dată se taie cifra unităţilor); tăierea ultimei cifre se obţine prin împărtirea întreagă la **10**.

Exemplu: **NrCifre(123)= 1+ NrCifre(12)= 1+(1+NrCifre(1))=1+1+1=3.**

```
type Natural = 0..MaxLongInt;

function  NrCifre(n:Natural) :Natural;
 begin
        if n in [0..9]
                then NrCifre:=1
                else NrCifre:=1 + NrCifre(n div 10);
end;
```

2. Determinarea sumei cifrelor a unui număr natural **n:**

- pentru **n ∈{0,1,2,3,4,5,6,7,8,9}** evident suma cifrelor este **n**;
- pentru **n ≥ 10**, se poate gândi că suma cifrelor lui n este ultima cifră + suma cifrelor numărului obţinut din **n**, din care am tăiat ultima cifră (de fiecare dată se taie cifra unităţilor); tăierea ultimei cifre se obţine prin împărtirea întreagă la **10** iar ultima cifră se poate obţine prin restul împărţirii întregi a lui **n** la **10**.

Exemplu: **SumaCifre(123)=3+SumaCifre(12)=3+(2+SumaCifre(1))=3+2+1=6**.

```
type Natural = 0..MaxLongInt;

function SumaCifre(n:Natural):Natural;

begin

        if n in [0..9]

                then SumaCifre:= n

                else SumaCifre:= n mod 10 + SumaCifre(n div 10);

end;
```

3. Determinarea produsului cifrelor a unui număr natural **n:**

- pentru **n∈ {0,1,2,3,4,5,6,7,8,9}** evident produsul cifrelor este **n**;
- pentru **n ≥ 10,** se poate gândi că produsul cifrelor lui n este ultima cifră * produsul cifrelor numărului obţinut din **n**, din care am tăiat ultima cifră (de fiecare dată se taie cifra unităţilor); tăierea ultimei cifre se obţine prin împărtirea întreagă la **10** iar ultima cifră se poate obţine prin restul împărţirii întregi a lui **n** la **10**.

Exemplu: **ProdusCifre(123)= 3* ProdusCifre(12)= 3*(2*ProdusCifre(1))=3*2*1=6.**

```
type Natural = 0..MaxLongInt;
function ProdusCifre(n:Natural):Natural;
begin
  if n in [0..9] then ProdusCifre:= n
                 else ProdusCifre:= n mod 10 * ProdusCifre(n div 10);
end;
```

4. Determinarea oglinditului numarărului natural **n** (oglinditul lui 123 este 321);

- pentru **n ∈{0,1,2,3,4,5,6,7,8,9}** evident oglinditul lui **n** este **n**;
- pentru **n ≥ 10**, se poate gândi că oglinditul lui n este ultima cifră * **$10^{NrCifre(n/10)}$ adunată cu oglinditul lui n / 10**

Exemplu: **Oglinda(123)=3*10^2+(Oglinda(12))=300+(2*10^1+Oglinda(1))=300+20+1=321.**

```
type Natural = 0..MaxLongInt;
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;
```

5. Determinarea cifrei de pe poziţia **poz** (de la dreapta la stânga) a unui număr natural **n.**

- se scad numărul de împărţiri ale lui **n** la **10** din variabila **poz**; dacă am ajuns la **0** se returnează (**n mod 10**) **.**

```
type Natural = 0..MaxLongInt;
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;
```

Concluzii

În general, algoritmii recursivi sînt recomandaţi în special pentru problemele ale căror rezultate sînt definite prin relaţii de recurenţă: analiza sintactică a textelor, prelucrarea structurilor dinamice de date, procesarea imaginilor ş.a.[4]

De asemenea, în comparaţie cu iterativitatea, recursivitatea este mai simplă din punct de vedere al structurii programului, și necesită mai puțin volum pentru scrierea programului, însă, aceasta necesită mai multă memorie și depanarea/testarea programelor este mai complicată. Totuși, în dependență de situație, algoritmii recursivi pot fi tranformați în iterative, și invers.

Bibliografie

1. https://ro.scribd.com/document/337119802/Iterativitatea
2. http://infoscience.3x.ro/c++/recursivitate.htm
3. https://www.codeit-project.eu/ro/differences-between-iterative-and-recursive-algorithms/
4. Manual de Informatică cl.11-a de Anatol Gremalschi, Editura: Știința, 2014
5. http://www.cs.ubbcluj.ro/~vcioban/InformaticaDidactica/RecursivitateaInProgramare.pdf
6. Manual de Informatică cl.9-a de Anatol Gremalschi, Editura: Știința, 2016