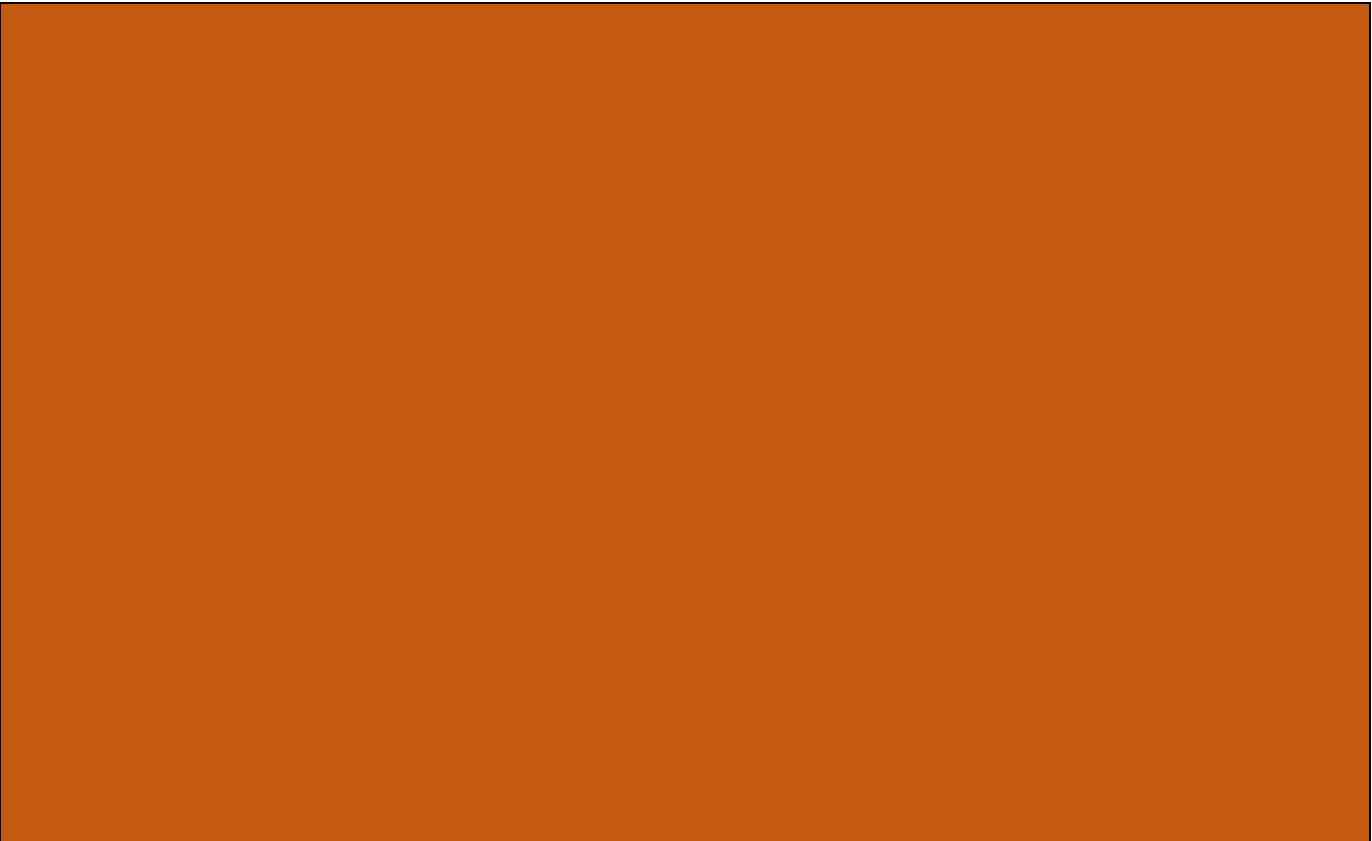


Iterativitatea și recursivitatea



Contents

Introducere	1
Iterativitatea vs Recursivitatea	2
Exemple de Program	3
Concluzie.....	6
Bibliografie.....	6

Introducere

Atât iterarea, cât și recursul se bazează pe o structură de control: Iterația folosește o structură de repetiție; recursiunea folosește o structură de selecție. Un algoritm iterativ va folosi instrucțiuni de buclă, cum ar fi pentru buclă, în timp ce buclă sau buclă pentru a repeta aceiași pași, în timp ce un algoritm recursiv, un modul (funcția) se recheamă din nou și din nou până când condiția de bază (condiția de oprire) este îndeplinită.

Un algoritm Iterativ va fi mai rapid decât algoritmul recursiv din cauza cheltuielilor generale cum ar fi funcțiile de apel și stivele de înregistrare în mod repetat. De multe ori, algoritmi recursivi nu sunt eficienți deoarece necesită mai mult spațiu și timp.

Algoritmi recursivi sunt utilizați în cea mai mare parte pentru a rezolva probleme complicate atunci când aplicarea lor este ușoară și eficientă. De exemplu, algoritmul Tower of Hanoi este ușor de recurs, în timp ce iterațiile sunt utilizate pe scară largă, eficiente și populare.

Iterativitatea vs Recursivitatea

- **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.

O problemă practică: tipul de ziua de naștere taie tortul de ziua și trebuie să se asigure că toată lumea din cameră primește o felie.

Soluția 1 – Iterativă : tipul care își serbează ziua de naștere taie tortul și se asigură ca toată lumea primește câte o felie.

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

Soluția 2 – Recursivă : Luați o felie de tort de pe tavă și treceți tava următoarei persoane care ia o felie din tavă și trece tava persoanei următoare, care ia o felie din tavă și trece tava la persoana următoare ...

Exemple de Program

Fiecare din următoarele exerciții vor fi rezolvate atât prin metoda iterativă cât și cea recursivă.

1. Calcularea sumei numerelor de la 1 până la **N**.

Metoda Iterativă :

```
program p1;
var n, sum, i: integer;

begin
  readln(n);

  for i:=1 to n do begin {Adunam numerele de la 1 la N pentru a afla}
    sum:=sum+i;           {suma numerelor}
  end;

  writeln(sum);
end.
```

Funcția Recursivă :

```
function sum(n: integer): integer;
begin
  if n=1 then sum:=1 else begin
    sum:=n+sum(n-1);           {Adaugam N la suma, apoi reapelam
                                {f-ctia cu N-1, adaugand nr-ul la}
  end;                          {suma, repetam procesul pana N=1}
end;
```

2. Calcularea produsului numerelor de la 1 la **N**.

Metoda Iterativă :

```
program p2;
var n, i: integer;
    produs: longint;

begin
  readln(n);

  produs:=1;

  for i:=1 to n do begin
    produs:=produs*i;          {Inmultim numerele de la 1 la N}
  end;

  writeln(produs);
end.
```

Funcția Recursivă :

```
function produs(n: integer): longint;
```

```

begin
  if n=1 then produs:=1 else begin {Inmultim produsul{cu valoarea 1}la N}
    produs:=n*produs(n-1);        {Reapelam f-ctia cu parametrul N-1}
  end;                             {Inmultind produsul}
end;

```

3. Calcularea sumei pătratelor numerelor de la 1 la N.

Metoda Iterativă :

```

program p3;
var n,i:integer;
    sum:longint;

begin
  readln(n);

  for i:=1 to n do begin {Adaugam la suma patratele nr-lor de la 1 la N}
    sum:=sum+(i*i);
  end;

  writeln(sum);
end.

```

Funcția Recursivă :

```

function suma_patratelor(n:integer):longint;
begin
  if n=1 then suma_patratelor:=1 else begin {Adaugam la suma patratul}
    suma_patratelor:=n*n+suma_patratelor(n-1);{numerelor de la N la 1}
  end;
end;

```

4. Calcularea sumei numerelor pare și a celor impare de la 1 la N.

Metoda Iterativă :

```

program p4;
var n,i:integer;
    sum_p,sum_i:integer;

begin
  readln(n);

  for i:=1 to n do begin {De la 1 la N}
    if i mod 2 = 0 then sum_p:=sum_p+i else {Determinam daca nr e par/impar}
      sum_i:=sum_i+i;                     {Adaugam nr-ul la suma respectiva}
    end;
  end;

```

```
writeln('suma nr-lor pare   : ',sum_p);
writeln('suma nr-lor impare : ',sum_i);
end.
```

Funcția Recursivă :

```
function sum(n:integer; var sum_i,sum_p:longint):longint;
begin
  if n=1 then begin
    sum_i:=sum_i+1;
  end else begin
    {In incinta f-ctie determinam}
    if n mod 2 = 0 then sum_p:=sum_p+n else {daca N e par/impar}
    sum_i:=sum_i+n; {Adaugam nr la suma respectiva}
    sum(n-1,sum_i,sum_p); {Reapelam f-ctia cu parametru N-1}
  end;
end;
```

5. Calcularea sumei numerelor de la 1 la **N ce sunt divizibile la numărul **X**.**

Metoda Iterativă :

```
program p5;
var x,n,i,sum:integer;

begin

  write('limit : '); readln(n);
  write('divisor : '); readln(x); {divizorul}

  for i:=1 to n do begin
    if i mod x = 0 then begin {Daca i e divizibil la X atunci}
      sum:=sum+i; {Suma multiplilor se maresta cu valoarea lui i}
    end;
  end;

  writeln(sum)

end.
```

Funcția Recursivă :

```
procedure sums(x:integer; divisor:integer; var sum:integer);
begin
```

```

if x=0 then sum:=sum+0 else begin
  if x mod divisor = 0 then begin
    sum:=sum+x;
    writeln(x);
    sums(x-1,divisor,sum);
  end else sums(x-1,divisor,sum);
end;
end;

```

{Daca **X** se imparte exact la divizor}
 {atunci prodecure se auto-apeleaza}
 {cu valoarea lui **X** scazuta cu 1}

Concluzie

Alcătuiind proiectul am ajuns la concluzia că în cea mai mare parte a cazurilor simple **Iterativitatea** se poate aplica, întrucât formularea programului este mai ușoară, însă în cazul în care programul are nevoie de un număr mai mare de iterări cu un set de condiții specifice, atunci **Recursia** este mai eficientă.

Bibliografie

1. <https://www.codeit-project.eu/ro/differences-between-iterative-and-recursive-algorithms/>