2020

Iterativitate sau Recursivitate

PATRICIA GORIUC

IPLT “SPIRU HARET”

Cuprins

1. Descrierea metodei/Aspecte teoretice 3
Iterativitatea 3
Recursivitatea 3
2. Probleme iterative explicate 4
3. Probleme recursive explicate 5
4. Concluzie. Iterativitatea vs Recursivitatea 7
5. Date bibliografice 7

1. Descrierea metodei/Aspecte teoretice

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 (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 structură iterativă se foloseşte este necesar ca numărul de iteraţii să fie finit.

Iteratia este execuţia repetată a unei porţiuni de program pană la indeplinirea unei condiţii (while,for etc.). Orice algoritm recursiv poate fi transcris intr-un algoritm iterativ si invers.

Recursivitatea

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 repetitive fără a utiliza cicluri. Mecanismul recursivitatii consta in posibilitatea ca un subprogram sa se autoapeleze.

Exista două tipuri de recursivitate:

I. recursivitate directă - cand un subprogram se autoapelează în corpul său ;
II. recursivitate indirectă - cînd avem două subprograme (x si y), iar x face apel la y şi invers

Pentru implementarea recursivităţii se foloseşte o zonă de memorie în care se poate face salvarea temporală a unor valori. La fiecare apel recursiv al unui subprogram se salvează în această zonă de memorie starea curentă a execuţiei sale.

Avantaje

- se realizează programe mai rapide;
- se evită operaţiile mult prea dese de;

- salvare pe stiva calculatorului;
- se evită incărcarea calculatorului în cazul apelurilor repetate.

Dezavantaje

- în cazul unui nr. mare de autoapelări, există posibilitatea ca segmentul de stiva să depaşească spaţiul alocat, caz în care programul se va termina cu eroare;
- recursivitatea presupune mai multă memorie în comparaţie cu iterativitatea.

2. Probleme iterative explicate

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

```
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.
```

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

```
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.
```

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

```
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.
```

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

```
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;

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

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

```
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 mareste cu valoarea lui
i}
    end;
  end;

writeln(sum)

end.
```

3. Probleme recursive explicate

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

```
      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
        end;                              {f-ctia cu N-1, adaugand nr-ul
la}
```

```
    end;                                    {suma, repetam procesul pana N=1}
```

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

```
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**.

```
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**.

```
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**.

```
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;                     {Daca X se imparte exact la
divizor}
        writeln(x);                     {atunci prodecure se auto-apeleaza}
        sums(x-1,divisor,sum);          {cu valoarea lui X scazuta cu 1}
      end else sums(x-1,divisor,sum);
    end;
  end;
```

4. Concluzie. Iterativitatea vs Recursivitatea

Nr.	Caracteristici	Iterativitate	Recursivitate
1	Necesarul de memorie	Mic	Mare
2	Tipul de executie	Acelasi	Acelasi
3	Structura programului	Complicata	Simpla
4	Volumul de munca	Mare	Mic
5	Testarea si depanarea	Simpla	Complicata

În principiu, orice algoritm poate fi elaborat atât recursiv cât şi iterativ. O funcţie recursivă, se reapelează de mai multe ori. Aceasta înseamnă un salt în program, de fiecare dată când se face un reapel, prin urmare consum de timp.

La fiecare reapel al funţiei pe segmentul de stivă alocat funcţiei, se crează un nou nivel, unde se memorează parametri şi variabilele locale. Acest lucru înseamnă consum de memorie, dar segmentul de stivă este oricum rezervat.

Observăm deci că o funcţie recursivă consumă mai mult timp, prin reapelări succeaive şi mai multă memorie, decât o funcţie iterativă. Acest lucru afectează programul doar în momentul în care se efectuează un număr foarte mare de reapelări ceea ce ar putea duce şi la ocuparea totală a segmentului de stivă, caz în care programul se intrerupe cu erpoare.

Pe de altă parte o funcţie recursivă se scrie cu mai multă uşurinţă, codul sursă poate fi mult restrâns, este mai naturală şi mai uşor de urmărit.

5. Date bibliografice

https://www.scribd.com/document/337119802/Iterativitatea

file:///C:/Users/Admin/Downloads/XI_Informatica%20(in%20limba%20romana).pdf

http://staff.cs.upt.ro/~ioana/sdaa/sda/l1.html