3/30/2020

Iterativitate sau recursivitate

Gheorghe Bîrsa

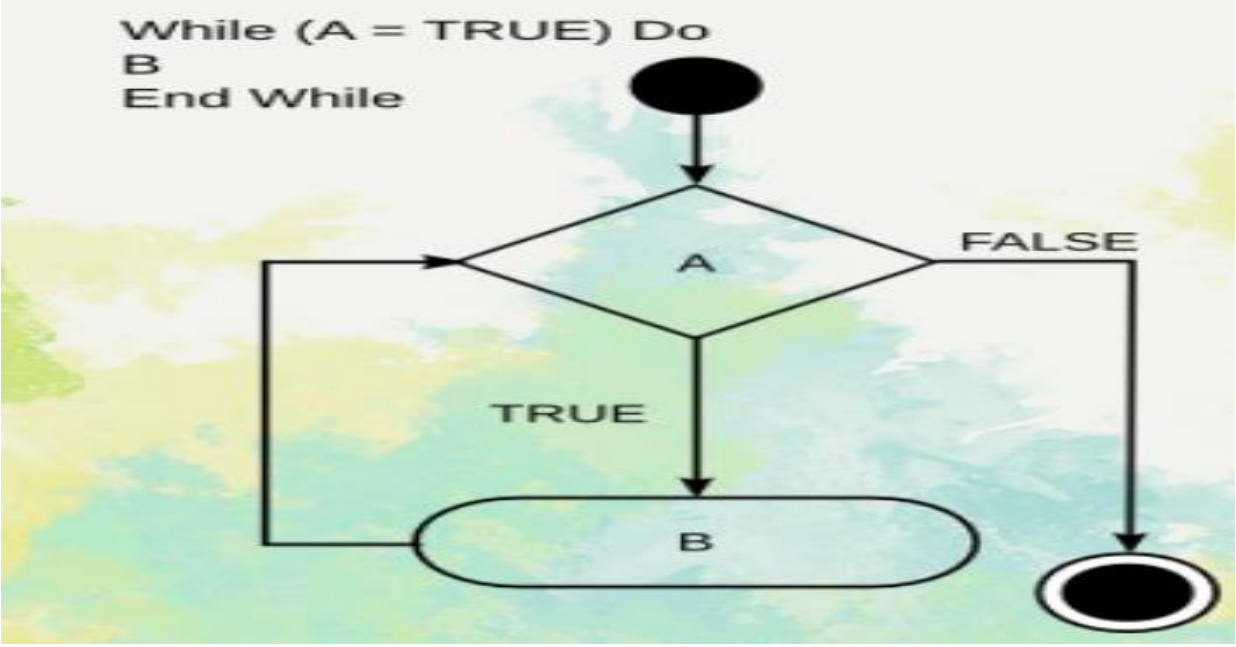
Cuprins

1.Descrierea metodei/Aspecte teoretice

1.1. Iterativitate

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.

Iteratia este execuţia repetată a unei porţiuni de program până la îndeplinirea unei condiţii("wile,for etc.) Dupa cum se stie, orice algoritm recursiv poate fi transcris intr-un algoritm iterativ si invers. [1]



[2]

1.2. Recursivitate

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).

Apelul recursiv al unei proceduri (functii) face ca pentru toti parametrii-valoare sa se creeze copii locale apelului curent (in stiva), acestea fiind referite si asupra lor facindu-se modificarile in timpul executiei curente a procedurii (functiei). Cind executia procedurii (functiei) se termina, copiile sint extrase din stiva, astfel incit modificarile operate asupra parametrilor-valoare nu afecteaza parametrii efectivi de apel, corespunzatori.

De asemenea pentru toate variabilele locale se rezerva spatiu la fiecare apel recursiv. [1]

Exista două tipuri de recursivitate:

1) recursivitate directă - cand un subprogram se autoapelează în corpul său ;

2) recursivitate indirectă - cînd avem două subprograme (x si y), iar x face apel la y şi invers ;

Se folosesc algoritmi recursivi atunci cînd calculele aferente sunt descrise în forma recursivă.

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

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

b)Subprogramul recursiv se va autoapela într-un mod în care se tinde spre ajungerea în situaţia de execuţie fără autoapel.

Pentru a permite apelarea recursivă a subprogramelor, limbajul Pascal dispune de mecanisme speciale de suspendare a execuţiei programului apelant, de salvare a informaţiei necesare şi de reactivare a programului suspendat .

Pentru implementarea recursivitaţii 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.

Variabilele locale ale subprogramului apelant au aceleaşi nume cu cele ale subprogramului apelat, orice referire la aceşti identificatori se asociază ultimului

set de valori alocate în zona de memorie. Zona de memorie rămîne alocată pe tot parcursul execuţiei subprogramului apelat şi se dealocă în momentul revenirii în programul apelat. Zona de memorie nu este gestionată explicit de programator ci de către limbaj.

La terminarea execuţiei subprogramului apelat recursiv, se reface contextul programului din care s-a facut apelul. Datorită faptului că la fiecare autoapel se ocupă o zonă de memorie, recursivitatea este eficientă numai dacă numărul de autoapelări nu este prea mare pentru a nu se ajunge la umplerea zonei de memorie alocată.

Recursivitatea oferă avantajul unor soluţii mai clare pentru probleme şi a unei lungimi mai mici a programului. Ea prezintă însă dezavantajul unui timp mai mare de execuţie si a unui spaţiu de memorie alocată mai mare. Este de preferat că atunci cînd programul recursiv poate fi transformat intr-unul iterativ să se facă apel la cel din urmă

Avantaje

-se realizează programe mai rapide;

-se evită operaţiile mult prea dese de;

salvare pe stiva calculatorului;

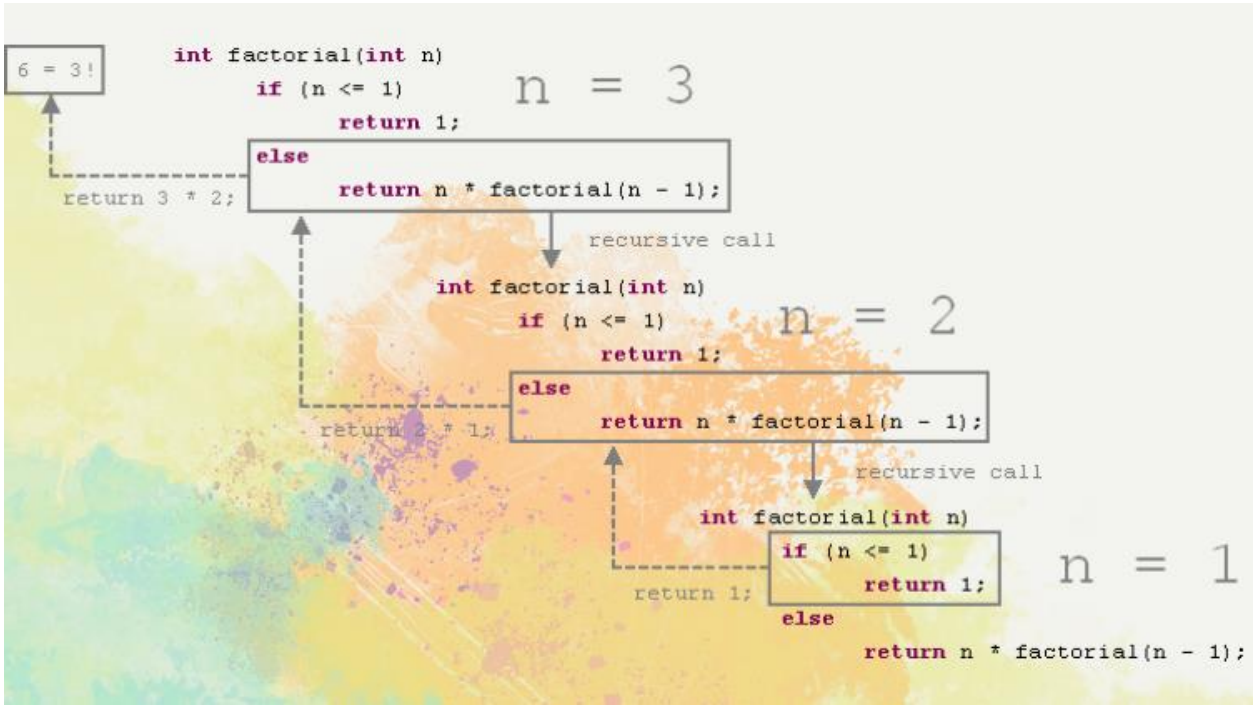
-se evită incărcarea calculatorului în

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.[3]



[2]

1.3. Studiu comparativ

Algoritmi recursivi vs iterativi

- **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.[4]

Nr.crtiteriu	Caracteristici	Iterativitate	Recursivitate
1.	Necesarul de memorie	mic	
2.	Timpul de executie	acelasi	
3.	Structura programului	complicata	simpla
4.	Volumul de munca necesar pentru scrierea programului	mare	mic
5.	Testarea si depanarea programelor	simpla	complicata

[5]

2.Probleme rezolvate

2.1. Probleme iterative

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.
```

2.2. Probleme recursive

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**.

```
unction 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;
```

3.Concluzie

In alegerea intre metoda recursiva si iterativa in elaborarea unui program trebuie sa se tina seama de :

▪ eficienta oferita programului de catre fiecare dintre variante,

▪ relatia dintre timpului de rulare si spatiului de memorie necesar

▪ nevoia de compactizare a programului.

Alegerea variantei recursive / iterative pentru scrierea unui program presupune:

▪ cunoasterea fiecarei metode in parte si a particularitatilor sale;

▪ cunoasterea tehnicii de transformare a recursivitatii in iteratie;

▪ studiu profund al structurilor de date optime reprezentarii datelor problemei;

▪ stapanirea tuturor facilitatilor oferite de limbajul de programare in care va fi implementat algoritmul.

Într-un final putem afirma că în majoritatea cazurilor simple ***Iterativitatea*** este aplicabilă, întrucât formularea programului este mai ușoară, însă în cazul că programul necesită un număr mare de iterări cu un set de condiții specifice, atunci ***Recursia*** este mai eficientă.

4.Bibliografie

https://prezi.com/qfmfcl_7jdpg/recursivitate-si-iterativitate/ [1]

https://prezi.com/hwzqekzxc5o9/iterativitate-sau-recursivitate/ [2]

https://ro.scribd.com/document/337119802/Iterativitatea [3]

https://www.codeit-project.eu/ro/differences-between-iterative-and-recursive-algorithms/ [4]

Manual de informatica clasa a 11-a Autor Anatol Gremalschi [5]