Iterativitate sau Recursivitate

Gutu Dinu

Cuprins

1.Recursivitate si Iterativitate

1.1 Recursivitate

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. Tehnicile în studiu se numesc respectiv recursia directă şi recursia indirectă şi au fost studiate în cadrul temei „Funcţii şi proceduri”.

În general, elaborarea unui program recursiv este posibilă numai atunci cînd se respectă următoarea regulă de consistenţă: soluţia problemei trebuie să fi e direct calculabilă ori calculabilă cu ajutorul unor valori direct calculabile. Cu alte cuvinte, defi nirea corectă a unui algoritm recursiv presupune că în procesul derulării calculelor trebuie să existe:

– cazuri elementare, care se rezolvă direct;

– cazuri care nu se rezolvă direct, însă procesul de calcul în mod obligatoriu progresează spre un caz elementar.

Recursivitatea e strins legata de iteratie, dar daca iteratia e executia repetata a unei portiuni de program, pina la indeplinirea unei conditii (while, repeat, for din PASCAL), 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.

Functia recursiva de calcul a factorialului:

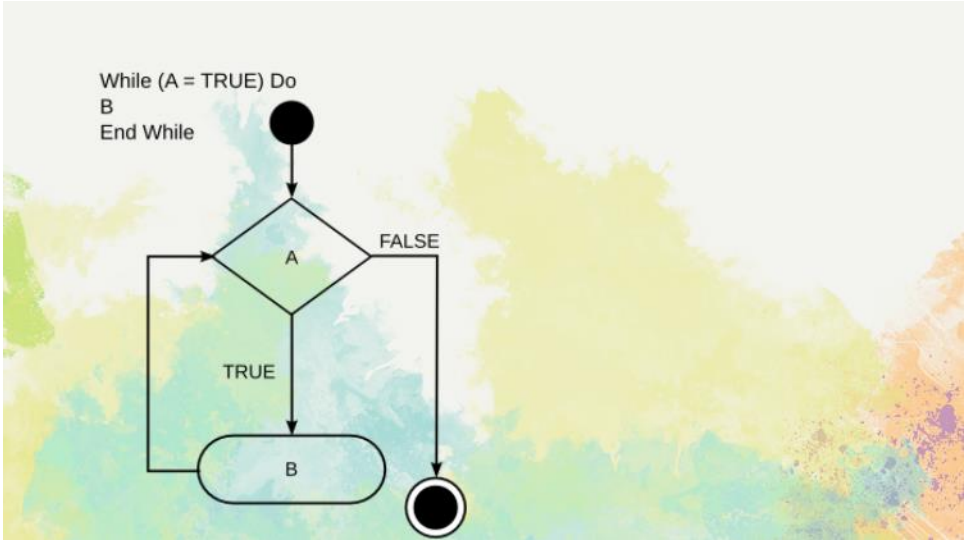
```
function fact(n:integer):integer;
    begin
        if n=1 then fact:=1
        else fact:=n*fact(n-1)
     end;
```

Demonstrarea corectitudinii cuprinde doi pasi:
-pentru n=1 valoarea 1 ce se atribuie factorialului este corecta
-pentru n>1, presupunind corecta valoarea calculata pentru predecesorul lui n de fact(n-1), prin inmultirea acesteia cu n se obtine valoarea corecta a factorialului lui n.

1.2 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 pană la indeplinirea unei condiţii(whiile,for etc.). Dupa cum sa vazut, orice algoritm recursiv poate fi transcris intr-un algoritm iterativ si invers.



1.3 Caracteristici

1.Necesarul de memorie:

Iterativitate: mic

Recursivitate: mare

2.Timpul de executie:

Acelasi in ambele cazuri

3.Structura programului:

Iterativitate: complicata

Recursivitate: simpla

4.Volumul de muncă necesar pentru scrierea programului:

Iterativitate: mare

Recursivitate: mic

5. Testarea şi depanarea programelor:

Iterativitate: simpla

Recursivitate: complicata

Iterativitate comparativ cu Recursivitate

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.

2.Probleme

2.1 Recursivitate

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 sumei pătratelor 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;
```

2.2 Iterativitate

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. Concluzie

După cum s-a văzut în capitolele precedente, orice algoritm recursiv poate fi transcris într-un algoritm iterativ şi invers. Alegerea tehnicii de programare – iterativitate sau recursivitate – ţine, de asemenea, de competenţa programatorului. Evident, această alegere trebuie făcută luînd în considerare avantajele şi neajunsurile fi ecărei metode, care variază de la caz la caz.

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/hwzqekzxc5o9/iterativitate-sau-recursivitate/

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

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

Manual de informatica clasa a 11-a Autor Anatol Gremalschi