Ciofu Viviana, 11C

Iterativitatea și recursivitatea

Cuprins

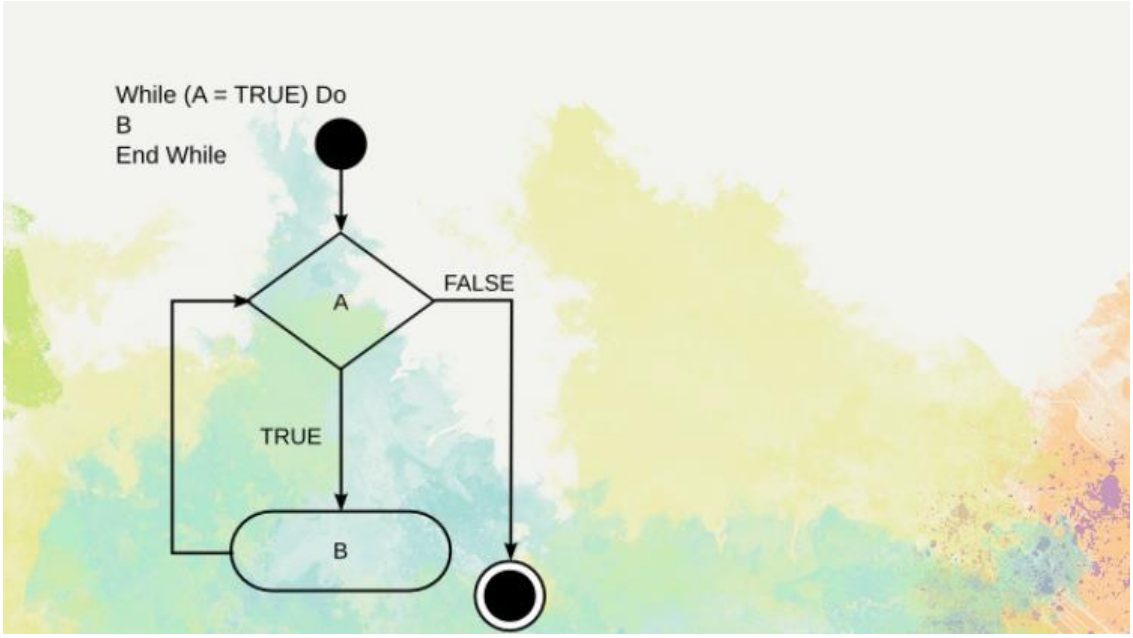
1.Recursivitate si Iterativitate

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.

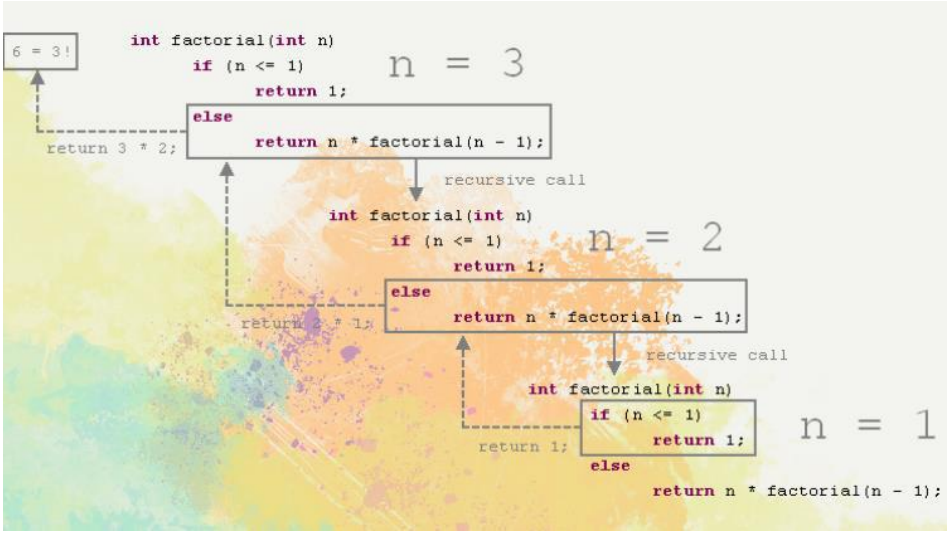
- Un algoritm Iterativ va fi mai rapid decât algoritmul recursiv din cauza cheltuielilor generale cum ar fi funcțiile de apel și stivele de înregistrare în mod repetat. De multe ori, algoritmii recursivi nu sunt eficienți deoarece necesită mai mult spațiu și timp.

- Iterația este execuția repetată a unei porțiuni de program pană la îndeplinirea unei condiții (whiile,for etc.). După cum s-a văzut, orice algoritm recursiv poate fi transcris într-un algoritm iterativ si invers.



1.2. 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 și recursia indirectă și au fost studiate în cadrul temei „Funcții și proceduri”.
- 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.
- Algoritmii recursivi sunt utilizați în cea mai mare parte pentru a rezolva probleme complicate atunci când aplicarea lor este ușoară și eficientă.



1.3 Caracteristici

- Necesarul de memorie
  Iterativitate: mic
  Recursivitate: mare

- Timpul de executie
  Acelasi in ambele cazuri

- Structura programului
  Iterativitate: complicata
  Recursivitate: simpla

- Volumul de muncă necesar pentru scrierea programului
  Iterativitate: mare
  Recursivitate: mic

- Testarea şi depanarea programelor
  Iterativitate: simpla
  Recursivitate: complicata

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.

2. Exemple de Probleme.

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

Iterație:

```
        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.
```

Recursie:

```
        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**.

Iterație:

```
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
```

Recursie:

```
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**.

Iterație:

```
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.
```

Recursie:

```
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**.

Iterație:

```
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.
```

Recursie:

```
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**.

Iterație:

```
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.
```

Recursie:

```
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

- Concluzionând, în urma celor analizate mai sus, putem afirma că iterativitatea este aplicabilă în cazurile mai simple, 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

- Manual clasa 11
- https://www.scribd.com/document/337119802/Iterativitatea
- https://prezi.com/hwzqekzxc5o9/iterativitate-sau-recursivitate/
- https://prezi.com/qfmfcl_7jdpg/recursivitate-si-iterativitate/
- https://www.codeit-project.eu/ro/differences-between-iterative-and-recursive-algorithms/