**Tema: Recursivitatea în programare**

**Subiectul principal:** Introducerea și aplicarea recursivității în rezolvarea problemelor de programare.

**Obiectivele de învățare**

Elevii vor învăța:

1. **Cunoștințe declarative**:
   * Ce este recursivitatea și cum funcționează.
   * Diferența dintre o funcție recursivă și o funcție iterativă.
   * Ce înseamnă „caz de bază” și „caz recursiv”.
   * Exemple de probleme care se pot rezolva eficient cu recursivitate (ex: factorialul unui număr, Fibonacci, parcurgerea arborelui binar).
2. **Cunoștințe procedurale**:
   * Cum se scrie o funcție recursivă.
   * Cum se identifică cazul de bază și cazul recursiv într-o problemă dată.
   * Cum se testează corectitudinea unei funcții recursive.
3. **Cunoștințe condiționale**:
   * În ce situații recursivitatea este o alegere mai bună decât iterația.
   * Care sunt dezavantajele recursivității (ex: consumul de memorie din cauza apelurilor succesive).
   * Cum să alegi între abordarea recursivă și cea iterativă în funcție de problema dată.

**Metode de predare și transmitere a cunoștințelor**

**1. Predarea cunoștințelor declarative**

* **Explicații teoretice**: Voi începe prin a defini conceptul de recursivitate folosind exemple simple și concrete. Voi prezenta diagrama apelurilor recursive pentru a ilustra fluxul funcțiilor.
* **Exemplu intuitiv**: „Recursivitatea este ca și cum te-ai uita într-o oglindă care reflectă o altă oglindă la nesfârșit.”
* **Probleme clasice**: Exemplul calculului factorialului unui număr:  
  n!=n⋅(n−1)!n! = n⋅(n-1)!n!=n⋅(n−1)! cu cazul de bază 0!=10! = 10!=1.

int factorial(int n) {

if (n == 0) return 1; // cazul de bază

return n \* factorial(n-1); // apel recursiv

}

* **Reprezentare grafică**: Desenez pe tablă schema apelurilor recursive pentru factorial, pentru a arăta cum funcționează „stack-ul apelurilor”.

**2. Dezvoltarea cunoștințelor procedurale**

* **Probleme aplicative**: Elevii vor rezolva împreună cu mine o problemă pas cu pas, începând cu identificarea cazului de bază și a cazului recursiv:  
  Exemplu: **Generarea termenilor din șirul Fibonacci**

int fibonacci(int n) {

if (n <= 1) return n; // cazurile de bază

return fibonacci(n-1) + fibonacci(n-2); // apeluri recursive

}

* **Activitate practică**: Elevii vor fi împărțiți în grupuri și vor implementa funcții recursive pentru:
  + Determinarea celui mai mare divizor comun (algoritmul lui Euclid).
  + Parcurgerea unui vector (sumarea elementelor folosind recursivitate).

**3. Dezvoltarea cunoștințelor condiționale**

* **Comparație recursivitate vs. iterație**: Rezolvăm aceeași problemă folosind și o funcție iterativă, analizând avantajele și dezavantajele fiecărei abordări (ex: recursivitate vs. buclă for).
* **Întrebări de reflecție**: „Ce se întâmplă dacă uităm cazul de bază?” (discutăm despre **apeluri infinite** și **stack overflow**).
* **Probleme complexe**: Discutăm situații în care recursivitatea este indispensabilă, cum ar fi **parcurgerea arborilor** (în algoritmi precum DFS - Depth First Search).

**Metode de evaluare**

Pentru a mă asigura că elevii au înțeles conceptele și le pot aplica corect, voi folosi:

1. **Evaluare formativă**:
   * Întrebări rapide în timpul lecției („Care este cazul de bază pentru această problemă?”).
   * Scenarii de tip „Ce se întâmplă dacă…” pentru a testa înțelegerea condițională.
2. **Temă practică**:
   * Implementați o funcție recursivă pentru suma cifrelor unui număr.
   * Implementați recursiv o problemă de parcurgere a unui arbore binar.
3. **Proiect în grup**:  
   Elevii vor crea un algoritm pentru rezolvarea unei probleme mai complexe folosind recursivitate, prezentând avantajele și dezavantajele soluției lor.
4. **Evaluare sumativă**:  
   Test scris cu exerciții teoretice (definirea recursivității, identificarea cazurilor de bază și recursive) și probleme practice (scrierea funcțiilor recursive).

**Argumentare**

**De ce va funcționa această metodă?**

1. **Explicațiile intuitive și exemplele simple**: Elevii vor înțelege recursivitatea prin analogii și exemple concrete, înainte de a aborda probleme complexe.
2. **Interactivitate**: Rezolvarea problemelor împreună cu mine și lucrul în grup vor stimula colaborarea și gândirea critică.
3. **Practică aplicativă**: Elevii vor scrie și testa codul, dezvoltând atât abilități procedurale, cât și înțelegere condițională.
4. **Evaluare continuă**: Verificarea progresului pe parcursul lecției și feedback-ul individual îi vor ajuta să își corecteze greșelile la timp.