

SUBPROGRAME

Un subprogram este un ansamblu ce poate conține tipuri de date, variabile și instrucțiuni destinate unei anumite prelucrări (calcul, citiri, scrieri). Subprogramul poate fi executat doar dacă este apelat de către un program sau un alt subprogram.

În limbajul Pascal, subprogramele sunt de 2 tipuri: funcții și proceduri. În C/C++, este utilizată doar noțiunea de funcție, o procedură din Pascal fiind asimilată în C/C++ unei funcții care returnează *void*.

O altă clasificare a subprogramelelor le împarte pe acestea în următoarele categorii:

- predefinite
- definite de către utilizator.

Dintre subprogramele predefinite, amintim subprogramele:

- matematice: *sin, cos, exp, log*;
- de citire/scriere: *read/write* (Pascal), *scanf/printf* (C/C++)
- de prelucrare a șirurilor de caractere: *substr, strlen, strcpy* (C/C++).

Exemplu: Conjectura lui Goldbach afirmă că orice număr par > 2 este suma a două numere prime. Să se scrie un subprogram care verifică acest lucru pentru toate numerele pare mai mici decât N dat.

Vom rezolva această problemă cu ajutorul subprogramelelor. Astfel, programul va conține:

- un subprogram *prim* care determină dacă un număr furnizat ca parametru este prim;
- un subprogram *verifica* prin care se obține dacă un număr furnizat ca parametru verifică proprietatea din enunț;
- programul principal, care apelează subprogramul *verifica* pentru toate numerele pare $2 < k < N$.

Soluția în C este următoarea:

```
#include "stdio.h"
#include "stdlib.h"

bool prim(int n)
{
    int i;
    for (i = 2; i <= n/2; i++)
        if(n%i==0) return false;
    return true;
}

bool verifica(int n)
```

```

{
    int i;
    for (i = 2; i<= n/2; i++)
        if (prim(i) && prim(n - i))
            return true;
    return false;
}

void main(void)
{
    //declarare variabile
    int n, k;
    //citire valori de intrare
    printf(„n=“);
    scanf(„%d“, &n);
    //prelucrare: verificare daca este indeplinita conditia
    //pentru fiecare k par
    for(k=4; k<=n; k+=2)
        if(!verifica(k))
        {
            printf(“%d nu verifica!\n“, k);
            //ieşire fara eroare
            exit(0);
        }
    //afisare rezultat
    printf(“proprietatea este indeplinita pentru numere >2 si
        <=%d\n“, n);
}

```

Observație: Problema a fost descompusă în altele mai mici. Rezolvarea unei probleme complexe este mai ușoară dacă descompunem problema în altele mai simple.

Avantaje ale utilizării subprogramelor:

- reutilizarea codului (un subprogram poate fi utilizat de mai multe subprograme);
- rezolvarea mai simplă a problemei, prin descompunerea ei în probleme mai simple, care conduc la elaborarea de algoritmi ce reprezintă soluții ale problemei inițiale;
- reducerea numărului de erori care pot apărea la scrierea programelor și despistarea cu ușurință a acestora.

Elementele unui subprogram sunt prezentate în continuare, cu referire la noțiunea de funcție din C/C++. Pentru funcțiile și procedurile din Pascal, aceste elemente se pot distinge în mod similar.

- Antetul funcției *prim* este: `bool prim(int n)`
- Numele funcției este `prim`
- Funcția *prim* are un parametru *n*.
- Funcția are un tip, care reprezintă tipul de date al rezultatului său. Tipul funcției *prim* este *bool*.

În cazul programelor C/C++, dacă tipul funcției este *void* înseamnă că funcția nu returnează un rezultat prin nume.

- Funcția are variabila proprie (locală) *i*.
- Funcția întoarce un anumit rezultat (în cazul funcției *prim*, o valoare booleană), prin intermediul instrucțiunii *return*.

Parametrii unui subprogram sunt de două tipuri:

- Parametri formali – cei ce se găsesc în antetul subprogramului;
- Parametri actuali (efectivi) – cei care se utilizează la apel.

De exemplu, în linia:

```
if(!verifica(k)) - parametrul k este un parametru efectiv.
```

Transmiterea parametrilor

Parametrii actuali trebuie să corespundă celor formali, ca număr și tip de date. Tipul parametrilor actuali trebuie fie să coincidă cu tipul parametrilor formali, fie să poată fi convertit implicit la tipul parametrilor formali.

- Pentru memorarea parametrilor, subprogramele folosesc segmentul de stivă, la fel ca pentru variabilele locale.
- Memorarea se face în ordinea în care parametrii apar în antet.
- În cadrul subprogramului, parametrii transmiși și memorați în stivă sunt variabile. Numele lor este cel din lista parametrilor formali.
- Variabilele obținute în urma memorării parametrilor transmiși sunt variabile locale.
- La revenirea în blocul apelant, conținutul variabilelor memorate în stivă se pierde.

Transmiterea parametrilor se poate realiza prin intermediul a două mecanisme:

- prin valoare
- prin referință.

Transmiterea prin valoare este utilizată atunci când dorim ca subprogramul să lucreze cu acea valoare, dar, în prelucrare, nu ne interesează ca parametrul actual (din blocul apelant) să rețină valoarea modificată în subprogram.

În toate apelurile din exemplul precedent, transmiterea parametrilor se realizează prin valoare.

Observație: Dacă nu ar exista decât acest tip de transmitere, ar fi totuși posibil să modificăm valoarea anumitor variabile care sunt declarate în blocul apelant. Acest lucru se realizează în situația în care am lucra cu variabile de tip pointer.

Exemplu: Să se scrie o funcție care interschimbă valorile a două variabile. Transmiterea parametrilor se va face prin valoare.

```
#include <iostream.h>
void interschimba(int *a, int *b)
{
    int aux = *a;
    *a = *b;
    *b = aux;
}

main()
{
    int x = 2, y = 3;
    interschimba(&x, &y);
    cout<< x << " " << y;
}
```

Observație: În limbajul C, acesta este singurul mijloc de transmitere a parametrilor.

Transmiterea prin valoare a tablourilor permite ca funcțiile să returneze noile valori ale acestora (care au fost modificate în funcții). Explicația este dată de faptul că numele tabloului este un pointer către componentele sale. Acest nume se transmite prin valoare, iar cu ajutorul său accesăm componentele tabloului.

Exemplu: Să se scrie o funcție care inițializează un vector transmis ca parametru.

```
#include <iostream.h>

void vector (int x[10])
{
    for (int i = 0; i < 10; i++)
        x[i] = i;
}

main()
{
    int a[10];
    vector(a);
    for (int i=0; i < 10; i++)
        cout << a[i] << " ";
}
```

Transmiterea prin referință este utilizată atunci când dorim ca la revenirea din subprogram variabila transmisă să rețină valoarea stabilită în timpul execuției programului.

În acest caz, parametrii actuali trebuie să fie referințe la variabile. La transmitere, subprogramul reține în stivă adresa variabilei.

La compilare, orice referință la o variabilă este tradusă în subprogram ca adresare indirectă. Acesta este motivul pentru care în subprogram putem adresa variabila normal (nu indirect), cu toate că, pentru o variabilă transmisă, se reține adresa ei.

Exemplu: Să se scrie o funcție care interschimbă valorile a două variabile.

```
#include <iostream.h>
void interschimba(int &a, int &b)
{
    int aux = a;
    a = b; b = aux;
}
main()
{
    int x = 2, y = 3;
    interschimba(x, y);
    cout<< x << " " << y;}
```

Supraîncărcarea funcțiilor

În C++ există posibilitatea ca mai multe funcții să poarte același nume. În această situație, funcțiile trebuie să fie diferite fie ca număr de parametri, fie ca tip. În acest din urmă caz, este necesară o condiție suplimentară: parametrii efectivi să nu poată fi convertiți implicit către cei formali.

Exemplu: Să se scrie o funcție supraîncărcată care afișează aria pătratului, respectiv a dreptunghiului, în funcție de numărul de parametri.

```
#include <iostream.h>
void arie(double latura)
{
    cout << "aria patratului este "<< latura * latura;
}

void arie(double lung, double lat)
{
    cout << "aria dreptunghiului este "<< lung * lat;
}

main()
{
    arie(3);    arie (4, 7);    double d = 7;    arie(&d);
}
```

Exerciții:

NOTĂ: Pentru toate funcțiile de mai jos, determinați și argumentați complexitatea lor.

1. Definiți următoarele 5 funcții:

- *read* – citește de la tastatură un tablou unidimensional;
- *positives* – returnează un tablou unidimensional alocat dinamic format din valorile strict pozitive dintr-un tablou unidimensional cu elemente de tip întreg, precum și numărul acestora;
- *multiply* – înmulțește cu x fiecare element al tabloului unidimensional v format din n numere întregi;
- *sort* – sortează crescător tabloul unidimensional v format din n numere întregi;
- *display* – afișează pe ecran tabloul unidimensional v format din n numere întregi.

Folosind doar apeluri utile ale celor 5 funcții definite anterior, scrieți un program care să afișează în ordine descrescătoare numerele strict negative dintr-un tablou unidimensional format din numere întregi.

2.

- Scrieți o funcție care să citească numărul de elemente ale unui tablou unidimensional, să aloce dinamic tabloul respectiv și apoi să citească valorile elementelor sale.
- Scrieți o funcție care să returneze poziția primului element strict mai mare decât un număr real x dintr-o secvență cuprinsă între doi indici i și j ($0 \leq i \leq j < n$) a unui tablou unidimensional format din n numere reale sau -1 dacă nu există nici un astfel de element.
- Scrieți o funcție care, folosind apeluri utile ale funcției definite anterior, afișează mesajul "*Da*" în cazul în care un tablou de numere reale, primit ca parametru, este sortat strict descrescător sau mesajul "*Nu*" în caz contrar.

3.

- Scrieți o funcție care să rearanjeze în ordine inversă elementele unei secvențe cuprinse între doi indici i și j ($0 \leq i \leq j < n$) dintr-un tablou unidimensional de numere întregi, fără a folosi un tablou suplimentar.
- Scrieți o funcție care, pentru un număr natural k , rearanjează elementele unui tablou unidimensional t format din n numere întregi $t[0], t[1], \dots, t[n-1]$ în ordinea $t[k+1], \dots, t[n-1], t[0], \dots, t[k]$ ($0 \leq k < n$), folosind doar apeluri utile ale funcției definite anterior.

4.

- Scrieți o funcție care să calculeze frecvența de apariție a unui număr întreg x într-o secvență cuprinsă între doi indici i și j ($0 \leq i \leq j < n$) dintr-un tablou unidimensional format din n numere întregi.
- Scrieți o funcție care, folosind apeluri utile ale funcției definite anterior, afișează mesajul "*Da*" în cazul în care un tablou unidimensional de numere întregi, primit ca parametru, are elemente distincte sau mesajul "*Nu*" în caz contrar.

- 5.
- Se da un sir de numere intregi. Se cauta un subsir cu lungimea cuprinsa intre l si u, format din elemente consecutive ale sirului initial, cu suma elementelor maxima.
- 6.
- Se considera un numar natural n. Determinati cel mai mic numar m care se poate obtine din n, prin eliminarea unui numar de k cifre din acesta fara a modifica ordinea cifrelor ramase.
- 7.
- Se citesc de la tastatura un numar natural n si un sir de numere naturale. Sa se scrie un program care aafiseaza indicii i si j care indeplinesc urmatoarele conditii:
 - a) $1 \leq i < j \leq n$;
 - b) $a_i > a_k, a_j > a_k, \forall k, i+1 \leq k \leq j-1$;
 - c) diferenta $j-i$ este maxima.

Probleme propuse:

- 8.
- Se considera N numere intregi care trebuie repartizate in p grupuri. Grupurile sunt identificate prin numere naturale cuprinse intre 1 si p. Repartizarea in grupuri trebuie sa se realizeze astfel incat suma numerelor din oricare grup i sa fie divizibila cu numarul total de numere care fac parte din grupurile identificate prin valori cuprinse intre i si p.
- 9.
- Consideram ca toate punctele de coordonate intregi din plan (coordonate mai mici de 1000) sunt colorate in negru, cu exceptia a n puncte care sunt colorate in rosu. Doua puncte rosii aflate pe aceeasi linie verticala (au aceeasi ordonata sau aceeasi abscisa) pot fi unite printr-un segment. Coloram in rosu toate punctele de coordonate intregi de pe acest segment. Repetam operatia cat timp se obtin puncte rosii noi. Cunoscand coordonatele celor n puncte care erau initial rosii, aflati numarul maxim de puncte rosii care vor exista in final.
- 10.
- Se considera o matrice binara de dimensiune m x n (elementele matricei sunt 0 sau 1). Se cere sa se determine aria maxima care poate fi acoperita de doua dreptunghiuri care contin numai elemente cu valoare 0 (dreptunghiurile nu se pot suprapune).

<i>in.txt</i>	<i>out.txt</i>
6 8	23
1 0 0 0 0 0 0 0	
1 0 0 0 0 0 0 0	
1 1 1 0 0 0 1 1	
0 0 1 0 0 0 1 1	
0 0 1 0 0 0 1 1	
0 0 1 1 1 1 1 1	

Recursivitate

1. *Tipuri de subprograme recursive*
2. Verificarea corectitudinii subprogramelor recursive
3. Complexitatea timp
4. Criterii de alegere; avantaje si dezavantaje
5. Greseli tipice in elaborarea subprogramelor recursive
6. Culegere de probleme

1. Tipuri de subprograme recursive

2. Verificarea corectitudinii subprogramelor recursive

3. Complexitatea timp

4. Criterii de alegere; avantaje si dezavantaje

5. Greseli tipice in elaborarea subprogramelor recursive

6. Culegere de probleme

(1) bazate pe funcții (primitiv) recursive unare, (funcția se apelează pe ea însăși în mod direct, ca în cazul calculării factorialului);

(2) bazate pe funcții (primitiv) recursive de mai multe argumente (ca în cazul calculării cmmdc pentru două numere naturale);

acestea sunt cunoscute sub numele de recursivitate liniară directă:

```
int cmmdc1 (int x, int y)
{ if (x==y) return x;
  else
    if (x>y) return cmmdc1(x-y, y);
    else return cmmdc1(x, y-x);
}
```

```
int cmmdc2 (int x, int y)
{ if (0==y) return x;
  else
    return cmmdc2(y, x%y);
}
```

1. **Tipuri de subprograme recursive**
 2. **Verificarea corectitudinii subprogramelor recursive**
 3. **Complexitatea timp**
 4. **Criterii de alegere; avantaje si dezavantaje**
 5. **Greseli tipice in elaborarea subprogramelor recursive**
 6. **Culegere de probleme**
-

(3) bazate pe funcții care se apelează una pe alta (așa numita recursivitate indirectă),

```
int a (int n)
{
    if (0==n) return 1;
    return a(n-1)+b(n-1);
}
int b (int n)
{
    if (0==n) return 1;
    return a(n-1)*b(n-1);
}
```

1. **Tipuri de subprograme recursive**
 2. **Verificarea corectitudinii subprogramelor recursive**
 3. **Criterii de alegere**
 4. **Avantaje si dezavantaje**
 5. **Greseli tipice in elaborarea subprogramelor recursive**
 6. **Culegere de probleme**
-

(4) bazate pe funcții care au nevoie de mai multe valori anterioare pentru a calcula valoarea curentă (așa numita recursivitatea neliniară sau în cascadă, ca în cazul determinării unui element al șirului lui Fibonacci după formula:

```
int fibonacci (int n)
{
    if (n<=1) return 1;
    return fibonacci(n-1) + fibonacci(n-2);
}
```

1. *Tipuri de subprograme recursive*
 2. **Verificarea corectitudinii subprogramelor recursive**
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

- se face intr-un mod similar verificării corectitudinii subprogramelor nerecursive.
- este mult simplificată de forma subprogramului (care permite utilizarea comodă a metodei inducției matematice complete):
 - se verifică mai întâi dacă toate cazurile particulare (de terminare a apelului recursiv) funcționează corect;
 - se trece apoi la o verificare formală, prin inducție, a funcției recursive corespunzătoare, pentru restul cazurilor.

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

Exemplificare: calculul factorialului unui număr

int factorial (int n)

```
{  
  if (0==n) return 1;  
  return n*factorial(n-1);  
}
```

Demonstrarea corectitudinii: doi pasi:

- pentru $n = 0$, valoarea 1 returnată de program este corectă;
- dacă $n > 1$ atunci, presupunând corectă valoarea returnată pentru $(n-1)$, prin înmulțirea acesteia cu n se obține valoarea corectă a factorialului numărului natural n , valoare returnată de subprogram.

În ambele situații este satisfăcută condiția de oprire.

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

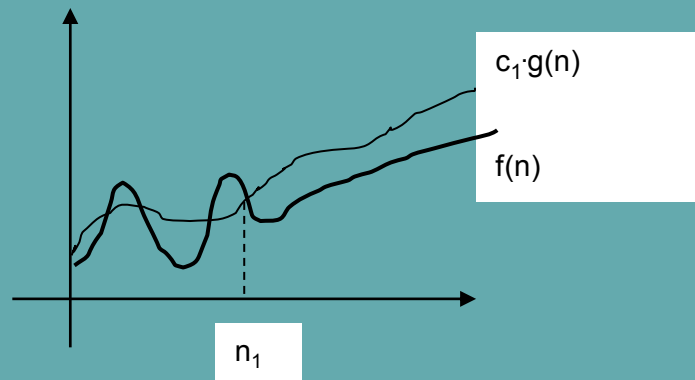
Exemplul 1

Fie $f : \mathbb{N} \rightarrow \mathbb{N}$, $f(n) = 5n^3 + 2n^2 + 22n + 6$;
spunem că f tinde asimptotic către n^3 și notăm acest lucru cu $O(n^3)$

Definiție 3

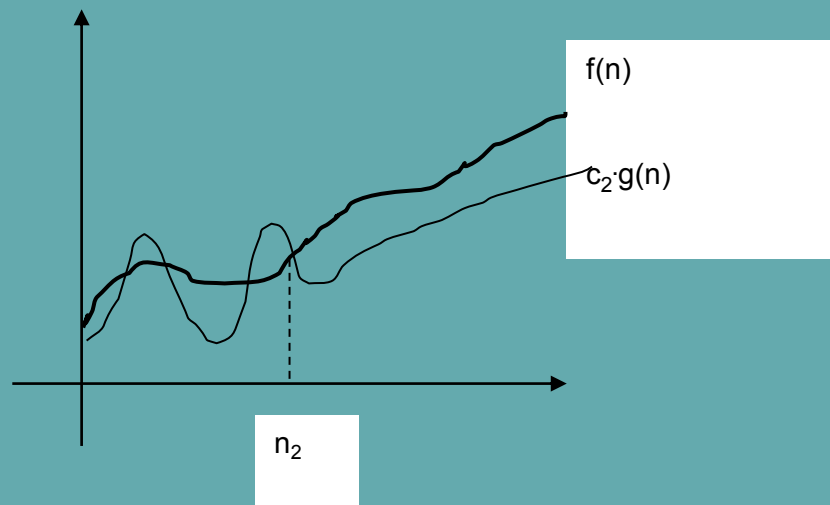
Fie $f, g : \mathbb{N} \rightarrow \mathbb{R}_+$

- (i) $f(n) = O(g(n))$ și citim “ $f(n)$ este de ordin cel mult $g(n)$ ” sau “ $f(n)$ este O mare de $g(n)$ ” \Leftrightarrow
(\exists) constantele $c_1 > 0$ și $n_1 \in \mathbb{N}$ astfel încât $f(n) \leq c_1 \cdot g(n)$, (\forall) $n \geq n_1$.



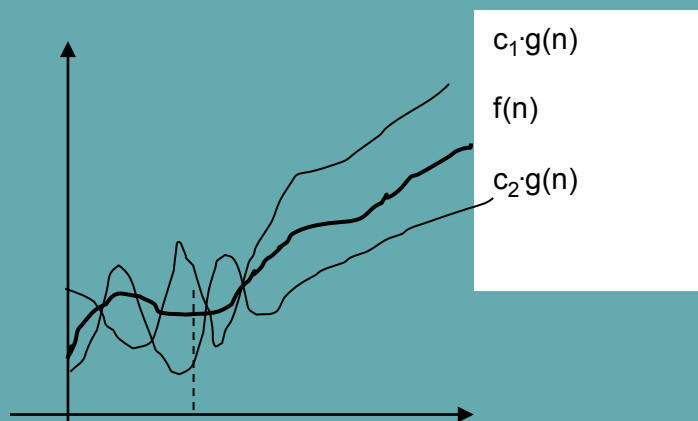
1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

(ii) $f(n) = \Omega(g(n))$ și citim “f(n) este de ordin cel puțin g(n)” sau “f(n) este omega mare de g(n)” \Leftrightarrow
(\exists) constantele $c_2 > 0$ și $n_2 \in \mathbb{N}$ astfel încât $f(n) \geq c_2 \cdot g(n)$, (\forall) $n \geq n_2$.



1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. **Complexitatea timp**
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

(iii) $f(n) = \Theta(g(n))$ și citim “ $f(n)$ este de ordin $g(n)$ ” sau “ $f(n)$ este theta de $g(n)$ ” \Leftrightarrow
 $f(n) = O(g(n))$ și $f(n) = \Omega(g(n))$.



Spunem că g este o limită asimptotică superioară,
o limită asimptotică inferioară, respectiv
o limită asimptotică pentru f .

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. **Complexitatea timp**
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

Exemplul 2

Revenim la notația O și la funcția polinomială

$$f(n) = 5n^3 + 2n^2 + 22n + 6 \Rightarrow$$

$f(n) = O(n^3)$, de exemplu, pentru $c_1 = 6$ și $n_1 = 10$;

$f(n) = O(n^4)$, de exemplu, pentru $c_1 = 1$ și $n_1 = 6$ sau
pentru $c_1 = 36$ și $n_1 = 1$;

$f(n) \neq O(n^2)$,

presupunem prin absurd că există $c_1 > 0$ și $n_1 \in \mathbb{N}$ astfel încât

$$5n^3 + 2n^2 + 22n + 6 \leq c_1 \cdot n^2, (\forall) n \geq n_1 \Leftrightarrow$$

$$5n^3 + (2 - c_1) \cdot n^2 + 22n + 6 \leq 0 \text{ etc.}$$

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. **Complexitatea timp**
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

Exemplul 3

Fie $f_1 : \mathbf{N} \rightarrow \mathbf{N}$, $f_1(n) = 3n \cdot \log_2 n + 5n \cdot \log_2(\log_2 n) + 2 \Rightarrow$

$f_1(n) = O(n \cdot \log n)$ pentru că $\log n$ domină $\log(\log n)$.

Analog, $f_2(n) = O(n^2) + O(n) \rightarrow f_2(n) = O(n^2)$

pentru că $O(n^2)$ domină $O(n)$.

Observația 1

A) Specificarea bazei logaritmilor nu este necesară ea intervenind cu cel mult un coeficient constant, conform formulei: $\log_a x = \frac{\log_b x}{\log_b a}$

B) Analog, nici specificarea bazei exponențialei nu este necesară pentru că:

$$\forall x > 0: x = 2^{\log_2 x} \rightarrow n^c = 2^{c \cdot \log_2 n} \Rightarrow$$

$2^{O(\log n)}$ este o limită superioară pentru n^c , unde c este o constantă oarecare.
Evident, și $2^{O(n)}$ este o limită superioară pentru n^c .

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

Observația 2

Limitele asimptotice de tipul n^c se numesc limite polinomiale.

Limitele asimptotice de tipul 2^{n^δ} se numesc limite exponențiale.

Limitele asimptotice de tipul $k \cdot n$ se numesc limite lineare.

Limitele asimptotice de tipul \sqrt{n} se numesc limite sublineare.

Observația 3

Pe lângă notațiile \mathbf{O} și $\mathbf{\Omega}$ mai există și notațiile \mathbf{o} și $\mathbf{\omega}$, obținute din Definiția 2 prin înlocuirea inegalității \leq cu inegalitatea strictă $<$, sau

$$f(n) = o(g(n)) \Leftrightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. **Complexitatea timp**
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

Exemplul 4

$$\sqrt{n} = o(n)$$

$$n = o(n \cdot \log \log n)$$

$$n \cdot \log \log n = o(n \cdot \log n)$$

$$n \cdot \log n = o(n^2)$$

$$n^2 = o(n^3)$$

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. **Complexitatea timp**
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

Propozitia 1

(i) Notațiile O , Ω , Θ , o , ω sunt tranzitive:

$$f(n) = O(g(n)) \ \& \ g(n) = O(h(n)) \ \rightarrow \ f(n) = O(h(n))$$

etc.;

(ii) Notațiile O , Ω , Θ , sunt reflexive, dar nu și o , ω

$$f(n) = O(f(n)) \ \text{dar} \ f(n) \neq o(f(n)) \ \text{etc.};$$

(iii) Notația Θ este simetrică dar nu și celelalte notații:

$$f(n) = \Theta(g(n)) \ \Leftrightarrow \ g(n) = \Theta(f(n)).$$

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. **Complexitatea timp**
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

Propozitia 2

Notațiile O , Ω etc. pot fi manipulate algebric, dar cu precautie (de ex. egalitatea din formula (5) are loc intr-un singur sens: de la stanga la dreapta):

1. $c \cdot O(f(n)) = O(f(n));$
2. $O(f(n)) + O(f(m)) = O(f(n));$
3. $O(O(f(n))) = O(f(n));$
4. $O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n));$
5. $O(f(n) \cdot g(n)) = f(n) \cdot O(g(n)).$

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. **Complexitatea timp**
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

Pentru a calcula complexitatea timp a unui algoritm (nr de pasi executati) vom proceda astfel:

- pentru fiecare etapa calculam:
 - nr de pasi necesari executarii ei,
 - de cate ori se executa etapa respectiva,
 - inmultim cele 2 valori;
- adunam valorile obtinute pentru etapele algoritmului si aplicam regulile calculului asimptotic.

	<i>Nr de pasi</i>	<i>Nr de executii</i>	<i>Total</i>
<i>Etapa nr. 1</i>	n^2	k	$k.n^2$
<i>Etapa nr. 2</i>	n^k	n	n^{k+1}
.....			

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. **Criterii de alegere; avantaje si dezavantaje**
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

Alegerea variantei recursive / iterative pentru scrierea unui program presupune:

- cunoasterea fiecărei 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.

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. **Criterii de alegere; avantaje si dezavantaje**
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

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.

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

- Exista o legatura stransa intre recursivitate si structurile de date de tip stiva, arbore, etc folosite in limbajele Borland Pascal si C++ pentru reprezentarea functiilor / procedurilor recursive (insasi definitia stivelor, arborilor, listelor realizandu-se recursiv).
- Iterativitatea minimizeaza complexitatea unor algoritmi
- Deseori, variantele iterative necesita folosirea explicita a structurii de tip stiva, generand astfel un cod extrem de laborios. In aceste situatii se considera solutia recursiva mai eleganta, datorita simplitatii sale.
- Recursivitatea poate fi inlocuita prin iteratie atunci cand recursivitatea este prea adanca sau cand limbajul de programare nu permite implementarea de apeluri recursive.
- Din punctul de vedere al memoriei solicitate, o varianta recursiva necesita un spatiu de stiva suplimentar pentru fiecare apel fata de varianta iterativa.
- Dimensiunea stivei trebuie aleasa astfel incat sa poata permite memorarea elementelor pentru toate iteratiile.

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

- Exemple de probleme pentru care solutia recursiva este mai putin intuitiva decat cea iterativa:
- Sa se calculeze suma cifrelor unui numar natural n .

```
// suma cifrelor unui numar - varianta
iterativa
#include<iostream.h>
#include<conio.h>
int suma(int n)
{ int s=0;
  while(n!=0)
  { s=s+n%10;
    n=n/10; }
  return s; }
void main()
{ int n;
  cout<<" n = "; cin>>n;
  int n1=n;
  cout<<" Suma cifrelor numarului
"<<n1<<" = "<< suma(n); }
```

```
➤ // suma cifrelor unui numar -
varianta recursiva
➤ #include<iostream.h>
➤ #include<conio.h>
➤ int suma(int n)
➤ { if(n==0) return 0;
➤   else return n%10 + suma(n/10); }
➤ void main()
➤ { int n;
➤   clrscr();
➤   cout<<" n = "; cin>>n;
➤   int n1=n;
➤   cout<<" Suma cifrelor numarului
"<<n1<<" = "<< suma(n); }
```

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

- Sa se verifice egalitatea a doua siruri de caractere introduse de la tastatura .

// egalitatea a 2 stringuri - varianta iterativa

- var a,b:string;
- egal:boolean;
- i:byte;
- begin
- writeln('introduceti sirurile :');
- readln(a); readln(b);
- egal:=true;
- if length(a)<>length(b) then egal:=false
- else
- for i:=1 to length(a) do
- if a[i]<>b[i] then egal:=false;
- if egal then writeln('sunt egale')
- else writeln('nu sunt egale')
- end.

// egalitatea a 2 stringuri - varianta recursiva

- var a,b:string;
- function egal(a,b:string):boolean;
- begin
- if length(a)<>length(b) then egal:=false
- else
- if a[1]<>b[1] then egal:=false
- else
- if length(a)=1 then egal:=true
- else
- egal:=egal(copy(a,2,length(a)-1),
- copy(b,2,length(b)-1))
- end;
- begin
- writeln('introduceti sirurile :');
- readln(a); readln(b);
- if egal(a,b) then writeln('sunt egale')
- else writeln('nu sunt egale')
- end.

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

- **Motivul:** Dificultatea descoperirii formulei de recurenta nu de iteratie
- Să se scrie un program care calculează suma

$$S_n = 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \dots + \frac{1}{2^n}$$

- Indicatie: In algoritmul recursiv se vor folosi 2 funcții definite astfel:

$$2^n = \begin{cases} 1 & , n = 0 \\ 2 \cdot 2^{n-1} & , n \geq 1 \end{cases} \quad S_n = \begin{cases} 1 & , n = 0 \\ \frac{1}{2^n} + S_{n-1} & , n \geq 1 \end{cases}$$

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. Culegere de probleme
-

// calculul sumei - varianta iterativa

```
> #include<iostream.h>
> #include<conio.h>
> unsigned n;
> void citire()
> {cout<<"n=";cin>>n;}
> float suma(unsigned n)
> {unsigned i;
>  long p=1;
>  float s=1;
>  for(i=1;i<=n;i++)
>      {p*=2;
>       s+=(float)1/p;}
>  return s;}
> void main()
> {clrscr();
>  citire();
>  cout<<"suma      pentru      n="<<n<<"
este="<<suma(n);
>  getch();}
```

// calculul sumei - varianta recursiva

```
> #include<iostream.h>
> #include<conio.h>
> unsigned n;
> void citire()
> {cout<<"n=";cin>>n;}
> long putere2(unsigned n)
> {if(n==0) return 1;
>  else return 2*putere2(n-1);}
> float suma(unsigned n)
> {if(n==0) return 1;
>  else
>  return (float)1/putere2(n)+suma(n-1);}
> void main()
> {clrscr();
>  citire();
>  cout<<"suma      pentru      n="<<n<<"
este="<<suma(n);
>  getch();}
```

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. **Greseli tipice in elaborarea subprogramelor recursive**
 6. Culegere de probleme
-

Greseli tipice in realizarea subprogramelor recursive:

1. Declararea globala a unor variabile care controleaza adresa de revenire (cazul cand apelurile se fac din interiorul unei structuri repetitive).
2. Declararea ca parametri transmisi prin valoare sau ca variabile locale a unor date structurate (de exemplu de tip tablou) micsoreaza semnificativ adancimea acceptata a recursivitatii, deoarece memorarea lor necesita un spatiu mare de memorie.

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. *Greseli tipice in elaborarea subprogramelor recursive*
 6. Culegere de probleme
-

4. Absenta unei conditii de oprire.
5. Exprimarea unei conditii de oprire in care nu intervin nici variabile locale si nici parametrii transmisi prin valoare sau prin referinta ai subprogramului.
6. Marirea dimensiunii problemei prin transmiterea in cadrul autoapelurilor a unor parametri actuali care nu tind sa se aproprie de valorile impuse prin conditia de oprire.
7. Neutilizarea directivei forward (limbajul Borland Pascal) in cazul declararii unor subprograme indirect recursive

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. *Culegere de probleme re rezolvate si propuse*
-

Probleme propuse:

1. Calculul recursiv al mediei aritmetice într-un vector.
2. Calculul recursiv al maximului și al minimului dintr-un vector.
3. Calculul recurent / inductiv al funcției:

$$f_n(x) = \begin{cases} \frac{1}{1+x}, n=0 \\ f_{n-1}(x) + \frac{2^n}{1+x^{2^n}}, n > 0 \end{cases}$$

4. Calculul recurent / inductiv al funcției:

$$f_{n,k} = \begin{cases} \frac{1}{k(k+1)}, n=1 \\ f_{n-1,k} + \frac{1}{(k+n-1)(k+n)}, n > 1 \end{cases} \quad \text{unde } k \geq 1 \text{ este un număr natural.}$$

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. *Culegere de probleme re rezolvate si propuse*
-

5. Valoarea funcției Ackerman în variantă recursivă și nerecursivă pentru perechea (m, n) , unde funcția este definită astfel:

$$Ack(m, n) = \begin{cases} n+1, m=0 \\ Ack(m-1, 1), n=0 \\ Ack(m-1, Ack(m, n-1)), \text{ altfel} \end{cases}$$

6. Să se realizeze parcurgerea arborilor binari (preordine, postordine, inordine), recursiv și iterativ.
7. Se consideră șirul a_1, a_2, \dots, a_n în progresie aritmetică (i.e. $\forall n \geq 2: a_n = (a_{n-1} + a_{n+1})/2$). Să se calculeze recursiv suma dată prin formula de recurență:

$$S_n : \begin{cases} S_1 = \frac{1}{\sqrt{a_1} + \sqrt{a_2}}, n=1 \\ S_n = S_{n-1} + \frac{1}{\sqrt{a_n} + \sqrt{a_{n+1}}}, n \geq 2 \end{cases}$$

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. *Culegere de probleme re rezolvate si propuse*
-

8. Se consideră șirul a_1, a_2, \dots, a_n in progresie geometrica (i.e. $\forall n \geq 2: a_n = \sqrt{a_{n-1} \cdot a_{n+1}}$). Să se calculeze recursiv suma dată prin formula de recurență:

$$S_n : \begin{cases} S_1 = \frac{\sqrt{a_1}}{\sqrt{a_2} - \sqrt{a_1}}, n = 1 \\ S_n = S_{n-1} + \frac{\sqrt{a_n}}{\sqrt{a_{n+1}} - \sqrt{a_n}}, n \geq 2 \end{cases}$$

9. Să se calculeze recursiv suma: $\frac{1}{a_1 a_2 \dots a_k} + \frac{1}{a_2 a_3 \dots a_{k+1}} + \dots + \frac{1}{a_n a_{n+1} \dots a_{n+k-1}}$

unde șirul $(a_n)_{n \geq 1}$ este reprezentat printr-o progresie aritmetică.

10. Să se calculeze recursiv suma

$a_1 a_2 \dots a_k + a_2 a_3 \dots a_{k+1} + \dots + a_{n+1} a_{n+2} \dots a_{n+k}$ unde șirul $(a_n)_{n \geq 1}$ este reprezentat printr-o progresie aritmetică.

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. *Culegere de probleme re rezolvate si propuse*
-

11. Se consideră matricea: $A = \begin{pmatrix} 1 & 1 & a \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$. Calculați recursiv A^n , $n \geq 2$.

12. Idem pentru matricele:

$$A = \begin{pmatrix} 0 & e^x & e^{-x} \\ e^{-x} & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad A = \begin{pmatrix} 0 & e^x + 1 & e^{-x} + 1 \\ e^{-x} & 0 & 1 \\ e^x & 0 & 0 \end{pmatrix}, \quad A = \begin{pmatrix} -a & a & a \\ a & -a & a \\ a & a & -a \end{pmatrix}.$$

13. Fiind dată matricea $A = \begin{pmatrix} x & 0 & x \\ 0 & y & 0 \\ x & 0 & x \end{pmatrix}$, să se calculeze $\sum_{k=1}^n A^k$.

14. Să se calculeze în variantă recursivă și iterativă primele n ($n \geq 5$ dat) elemente din șirul lui Fibonacci.
15. Problema turnurilor din Hanoi.
16. Să se calculeze recursiv a^n , $n \geq 2$.

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. *Culegere de probleme re rezolvate si propuse*
-

19. Să se calculeze recursiv C_n^k , $n \geq 2$, $0 \leq k \leq n$.
20. Să se scrie funcția recursivă pentru calculul sumei cifrelor unui număr natural.
21. Să se scrie funcția recursivă care citește un număr oarecare de caractere și le afișează în ordinea inversă citirii. Atenție! nu se lucrează cu șiruri, nu se cunoaște numărul de caractere, iar sfârșitul șirului este dat de citirea caracterului '0'.
22. Se cere calculul recursiv al sumei a n numere naturale citite.
23. Să se realizeze programele recursive pentru problema aflării celui mai mare divizor comun pentru calculul simplu, dar și pentru calculul folosind algoritmul lui Euclid.

1. Tipuri de subprograme recursive
 2. Verificarea corectitudinii subprogramelor recursive
 3. Complexitatea timp
 4. Criterii de alegere; avantaje si dezavantaje
 5. Greseli tipice in elaborarea subprogramelor recursive
 6. *Culegere de probleme re rezolvate si propuse*
-

24. Să se caute rădăcina unei funcții crescătoare, cunoscându-se următorul rezultat: Fie f o funcție crescătoare. Dacă $f(a) < 0$ și $f(b) > 0$, atunci f are rădăcină în intervalul $[a, b]$.
25. Să se scrie programul C/C++ pentru realizarea căutării binare (recursiv și iterativ) (se caută cheia v într-un tablou sortat și se returnează indicele).
26. Drum minim. Între n orașe există o rețea de drumuri care permite ca dintr-un oraș să se ajungă în oricare dintre celelalte. Între două orașe, există cel mult un drum direct, de lungime cunoscută, iar timpul necesar parcurgerii unui drum este proporțional cu lungimea sa. Să se scrie programul recursiv pentru determinarea traseului pe care se poate merge între două orașe date, într-un timp minim.