

# Noțiuni introductive

- informații  $\rightarrow$  date  $\rightarrow$   $\rightarrow$  structuri de date (SD).
- sistemele de calcul ocupă mult timp cu
  - stocarea datelor (S)
  - accesarea datelor (A)
  - manipularea datelor (M)
- în toate domeniile (sisteme de operare, baze de date, compilatoare, grafică, inteligența artificială, etc) se pune problema (S)+(A)+(M)
- reprezentarea obiectelor din lumea reală în aplicații software necesită
  1. obiecte din lumea reală  $\xrightarrow{\text{MODELARE}}$  entități matematice
  2. mulțimea de operații definite pe entitățile de la 1.
  3. maniera în care entitățile de la 1. sunt **reprezentate** și **stocate** în memoria calculatorului
  4. algoritmi utilizați pentru a efectua operațiile de la 2.
    - (1)+(2)  $\rightarrow$  TAD
    - (3)+(4)  $\rightarrow$  SD

## Tipuri Abstracte de Date

“În dezvoltarea înțelegerii fenomenelor complexe, cel mai puternic mecanism disponibil omului este **abstractizarea**. Abstractizarea derivă din recunoașterea similarităților dintre anumite obiecte, situații sau procese din lumea reală precum și din decizia de a concentra aceste similarități și de a ignora, pentru moment, diferențele dintre ele.”

- C.A.R. Hoare

- Tip de date
  - *domeniu*
  - *operații*
- Un Tip Abstract de Date (TAD) este un tip de date cu următoarele proprietăți:

1. *specificarea obiectelor* din domeniu este independentă de reprezentarea lor;
  2. *specificarea operațiilor* este independentă de implementarea lor.
- Mulțimea operațiilor Tipului Abstract de Date definesc interfața (contractul) său.
  - un TAD este o entitate specificată matematic, care definește o mulțime – a instanțelor sale, având o *interfață* specifică – o colecție de *signaturi* ale operațiilor care pot fi aplicate pe o instanță a tipului de dată, pentru fiecare operație asigurându-se o specificație clară.
  - Domeniul unui tip (abstract) de dată poate fi definit fie prin enumerarea elementelor sale, în cazul în care este finit, fie printr-o regulă care descrie elementele sale.
  - După definirea domeniului unui tip (abstract) de date, este necesară *specificarea operațiilor* (date, rezultate, condiții, postcondiții).
  - “De ce abstractizare?”
    - Deoarece se impune specificarea operațiilor structurilor de date și amânarea detaliilor de implementare.
    - Există multe Tipuri Abstracte de Date, ca urmare decizia de a alege TAD-ul potrivit este un pas foarte important în proiectare.
    - Limbajele de nivel înalt furnizează deseori implementări pentru TAD-uri predefinite (biblioteca STL din C++, bibliotecile standard din Python, Java).
    - TAD de tip *container*: *vector* (**array**), *colecție* (**collection**), *mulțime* (**set**), *listă* (**list**), *dicționar* (**dictionary**, **map**), etc.
      - \* “De ce să încercăm să creem propriile noastre implementări pentru diferite TAD dacă unele limbaje ne oferă implementări ale TAD pe care ni-l dorim?”
        1. pentru a învăța modul de utilizare și creare de implementări de Tipuri Abstracte de Date.
        2. pentru a învăța comportamentul unora dintre cele mai utilizate Tipuri Abstracte de Date.

## Interfața unui TAD

Tipurile de operații din interfața unui TAD sunt următoarele:

- - operații care să permită crearea elementelor de acel tip (*constructori*);
- - operație care să permită distrugerea elementelor de acel tip (*destructor*);
- - operații de accesare a componentelor instanțelor;
- - operații specifice de manipulare a instanțelor;
- - operații de verificare ale unor proprietăți ale instanțelor;
- - operații specifice tipului de date.

## Folosirea abstractizării și încapsulării datelor în proiectarea programelor

1. **Încapsularea datelor** (ascunderea informației) este definită ca fiind ascunderea detaliilor de implementare ale unui obiect față de lumea exterioară (aplicațiile care îl folosesc).
2. **Abstractizarea** datelor este definită ca fiind separarea dintre *specificarea* unui obiect și *implementarea* lui.

În procesul de proiectare a programelor, folosirea abstractizării și încapsulării datelor sunt deosebit de importante, din următoarele motive:

- **Simplificarea procesului de dezvoltare** a programelor - dacă în faza de proiectare a programului am identificat că avem nevoie de tipurile de date A, B, C, ... cu acestea se poate lucra independent, deoarece nu interesează decât specificațiile lor.
- **Testarea și depanarea** se poate face mai ușor, deoarece fiecare dintre tipurile A, B, C, ... pot fi testate și verificate separat.
- **Reutilizarea** - Extragerea unei structuri de date dintr-o aplicație și folosirea acesteia în altă aplicație va fi deosebit de facilă în cazul în care pentru tipul respectiv de date am folosit încapsularea datelor.
- **Schimbarea reprezentării** unui tip de dată - se poate schimba reprezentarea unui tip de date fără a fi afectate programele care folosesc acel tip de date cu condiția ca operațiile tipului să nu fie modificate (interfața să rămână aceeași).

Una din cerințele esențiale ale programării prin abstractizarea și încapsularea datelor este să nu se expună în exterior (într-o aplicație) reprezentarea unui Tip Abstract de Date.

# Structuri de Date

- Domeniul *Structurilor de Date* (SD) se ocupă cu stocarea și accesarea datelor.
- În rezolvarea de probleme se manipulează informații. În organizarea datelor, conform algoritmului de rezolvare, informațiile de diferite tipuri se grupează în structuri, numite *structuri de date*.
- O SD putem să o considerăm din două puncte de vedere:
  1. **Logic**, ca și definiție (elementele ei constitutive și legăturile dintre ele).
  2. **Fizic**, ca mod de memorare (stocare).
    - (a) Structuri de date **statice**. O structură **statică** ocupă în memoria calculatorului o zonă de dimensiune constantă, în care elementele ocupă tot timpul execuției același loc. Exemple de structuri de date statice sunt: articolele, tablourile.
    - (b) Structuri de date **semistatice**. O structură **semistatică** ocupă în memoria calculatorului o zonă de dimensiune constantă, dar elementele ocupă loc variabil în timpul execuției programului. Exemple de structuri de date semistatice sunt tabele de dispersie.
    - (c) Structuri de date **dinamice**. O structură **dinamică** ocupă în memoria calculatorului o zonă care se alocă dinamic, în timpul execuției programului, pe măsura nevoilor de prelucrare, fără a avea o dimensiune constantă. Exemple de structuri de date dinamice sunt: liste înlănțuite, arbori.
- O structură de date logică ar putea fi implementată atât ca structură statică, semistatică sau dinamică. De exemplu, lista înlănțuită, care din punct de vedere conceptual este o structură dinamică (se modifică în timp), ar putea fi implementată ca o structură semistatică.
- Pentru un TAD vom discuta mai multe structuri de date folosite pentru implementarea acestuia, fiecare aducând beneficii specifice, dar și eventuale dezavantaje
  - În unele cazuri nu se poate spune foarte clar care ar fi cea mai “bună” structură de date pentru implementarea unui TAD.
  - Alegerea structurii de date corespunzătoare rezolvării unei probleme este extrem de importantă.
    - \* alegerea unor tehnici de structuri de date corespunzătoare poate avea ca efect reduceri masive de timp - analiza complexității operațiilor este importantă.
  - Cum putem proiecta SD noi?

TAD/Container	SD
<b>Vector Dinamic</b> <b>Matrice</b> <b>Colecție</b> <b>Mulțime</b> <b>Dicționare</b> Dicționar, Dicționar ordonat, Multidicționar, Multidicționar ordonat <b>Lista</b> <b>Lista ordonată</b> <b>Coadă</b> <b>Stiva</b> <b>Coadă cu priorități</b> <b>Arbore (binar)</b>	<b>tablou</b> <b>lista înlănțuită</b> - simplu - dublu - alocare dinamică - înlănțuiri pe tablou <b>tabela de dispersie</b> - liste independente - liste întrepătrunse - adresare deschisă <b>ansamblu</b> <b>arbore binar de căutare</b> - arbore echilibrat (AVL)