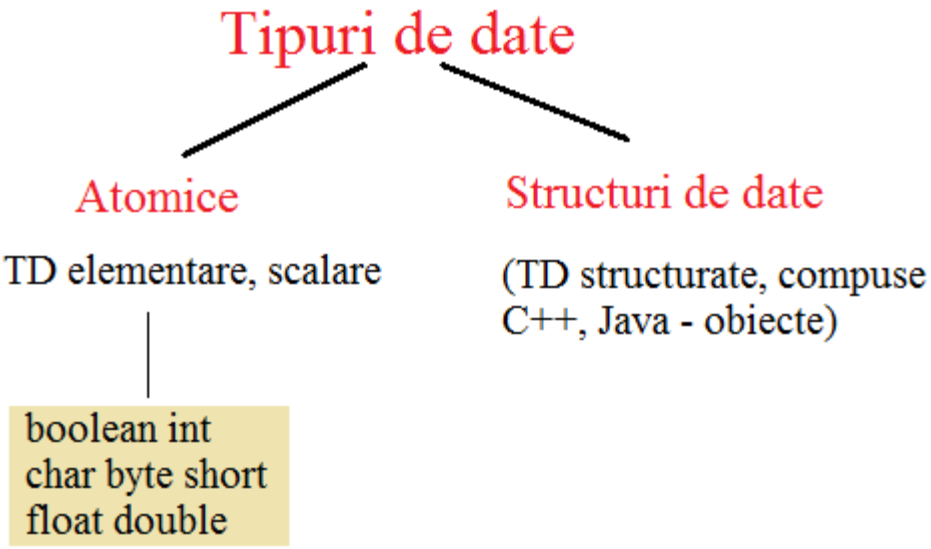
3. Tipuri de date

Un tip de data este caracterizat de:

- O mulţime de date (valori ϵ domeniului)
- O mulţime de operaţii
- Un identificator

Exemplu:

Tipul de dată - *Număr întreg* (‚Integer‘):
Un număr întreg (ϵ Z) între o limită superioară şi una inferioară şi un număr de operaţii:
– **+, -, *, /, %**
– în funcţie de definiţie şi **<, >, ==, sign, abs, odd, even, …**


Tipuri de date
Atomice
Structuri de date
TD elementare, scalare
(TD structurate, compuse
C++, Java - obiecte)
boolean int
char byte short
float double

Structurile de date pot fi:

- *Omogene*: toate componentele au acelaşi tip de dată
- *Heterogene*: componentele au tipuri diferite de dată

O **structură de date** este modul în care data este reprezentată în interiorul masinii, în memorie sau pe disc (cursul 1, vezi și cursul BD)

Definiţie 2. Structură de date

O structură de date este specificaţia pentru organizarea şi memorarea datelor, astfel încât să dea posibilitatea unui acces *eficient* pentru anumite clase de aplicaţii.

Cuprinde două aspecte esenţiale:

- Interfaţa: stabilirea operaţiilor posibile şi a comportamentului ca:
  - specificaţie abstractă (**T**ip **A**bstract de **D**ate TAD/ADT)
  - sau interfaţă concretă de programare (de exemplu biblioteci ca STL Standard Template Library din C++)
- Implementarea: *transpunerea (punerea în aplicare) concretă* într-un limbaj de programare prin structuri de memorie şi algoritmi cât mai eficienţi

Exemple:

Listele cu grupele de studenţi

Lista nodurilor şi arcelor în Modelele BREP

Inventarul unei figuri dintr-un joc pe calculator ca **mulţime** de obiecte

Arborii de directoare pentru gestionarea fişierelor

Reţeaua de străzi ca şi **graf** pentru planificatorul de rută

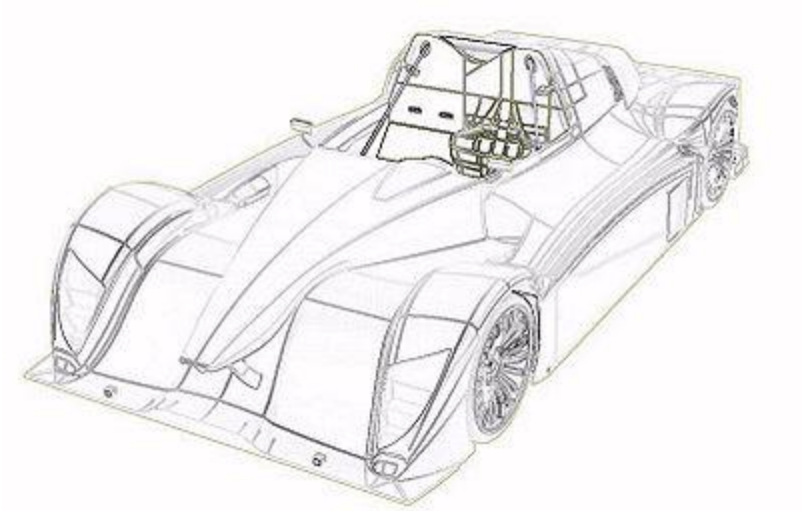
Cozi de aşteptare ale proceselor la gestionarea proceselor de către sistemul de operare

Stiva programului pentru gestionarea datelor în timpul execuţiei programului

B-**Arbori** ca indecşi pentru acces rapid într-un sistem de baze de date

Boundary representation

From Wikipedia, the free encyclopedia



In solid modeling and computer-aided design, **boundary representation**—often abbreviated as **B-rep** or **BREP**—is a method for representing shapes using the limits. A solid is represented as a collection of connected surface elements, the boundary between solid and non-solid.

Overview

Boundary representation models are composed of two parts: topology and geometry (surfaces, curves and points). The main topological items are: *faces*, *edges* and *vertices*. A face is a bounded portion of a surface; an edge is a bounded piece of a curve and a vertex lies at a point. Other elements are the *shell* (a set of connected faces), the *loop* (a circuit of edges bounding a face) and *loop-edge links* (also known as *winged edge links* or *half-edges*) which are used to create the edge circuits. The edges are like the edges of a table, bounding a surface portion.

TAD ca metodă independentă de specificare a interfeţei şi a semanticii

La TAD se aplică 2 principii:

- Încapsularea: Fiecare TAD are o interfaţă. Accesul la SD are loc exclusiv prin intermediul interfeţei (de exemplu: adaugaElement(…), stergeElement(….)).

- Abstractizarea/Principiul ascunderii detaliilor: Realizarea internă a unui modul TAD implementat rămâne ascunsă utilizatorului

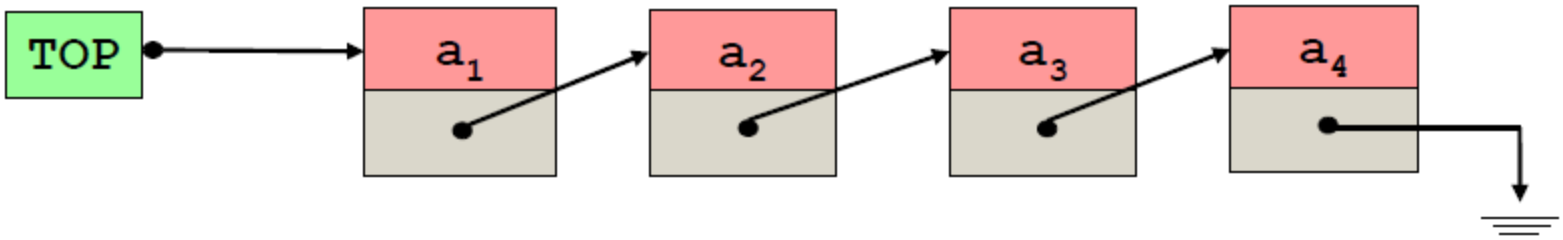
Specificaţia unui TAD descrie cum acţionează operaţiile asupra datelor, dar nu şi cum sunt reprezentate intern datele şi nici cum sunt implementate operaţiile

Abstractizare şi încapsulare - motive:

- Simplificarea procesului de dezvoltare a programelor în faza de proiectare: este suficient să ştiu cu ce date lucrez: D1,D2,D3,…, doar specificaţiile lor .
- Testarea şi depanarea mai simplă, fiecare dată se poate testa şi verifica separat.
- Reutilizare - Extragerea unei SD dintr-o aplicaţie şi folosirea ei în altă aplicaţie.
- Schimbarea reprezentării unui tip de dată - se poate fără a afecta programele care îl folosesc cu condiţia ca interfaţa să rămână aceeaşi (operaţiile tipului să rămână aceleaşi).

Poate exista una sau mai multe implementări ale unui TAD care au la bază concepte diferite.

De exemplu TAD “Listă” cu structura de date vector dinamic sau listă înlănţuită



Implementarea unui TAD

- Descrie reprezentarea internă a datelor şi
- Implementarea exactă a operaţiilor

Diferite implementări ale aceleaşi specificaţii de TAD ne dau posibilitatea să optimizăm performanţa
⇒Baza pentru argumentarea eficienţei ADT-ului

Exemplu:
Operaţia push(stack s, int e) implementată ca array:

```
void push (stack s, int e) {
   s.top = s.top + 1;
   s[s.top] = e;
}
```

Eficienţa unei implementări de TAD este hotărâtoare:

- Complexitatea de timp a operaţiilor
  - Inserarea unui element;
  - Ştergerea unui element;
  - Căutarea unui element

- Complexitatea de spaţiu a reprezentării interne a datelor

De obicei se face un compromis între eficienţa de spaţiu şi de timp: operaţiile mai rapide necesită de obicei spaţiu suplimentar, iar reprezentările mai compacte conduc la operaţii mai lente

CONTAINERUL

Definiţie: O SD (un obiect) de bază care conţine o colecţie de elemente (obiecte) şi are metode proprii pentru accesul la elemente.

- Container omogen:
  containerul conţine elemente de acelaşi tip
- Container eterogen:
  containerul care conţine elemente de tipuri diferite
- Container secvenţial:
  containerul ale cărui elemente sunt aranjate după o regulă (de ex. după index)

Operatii:

- *Creare container (vid)*
- *Inserare element*
- *Cautare element*
- *Stergere element*
- *Numarul elementelor*

CONTAINERUL

In funcţie de aplicaţie, cerinţe foarte diferite:

- Obiecte duplicate
- Ordine
- Acces poziţional
- Acces asociativ
- Acces iterativ

Tip colecţie	**Dinamic**	**Duplicate**	**Ordine**	**Acces**
Array/tablou	da/nu	da	da	poziţie
Set/mulţime	da	nu	nu	
Bag/multiset	da	da	nu	
List/listă	da	da	da	poziţie
Map, Hash Table	da	da	nu	asociativ (cheie)

ITERATOR PENTRU CONTAINER

Definiţie: Iteratorul este o SD de bază care permite traversarea un container, indiferent de implementare.

Containerul trebuie să furnizeze un mecanism de accesare a elementelor sale cu ajutorul iteratorului

Iteratorul conţine

- o referintă spre conteiner
- o referintă spre elementul *curent*

Tipuri de iteratori

- Iterator înainte: (se *incrementează, spre ultimul obiect al containerului)*
- Iterator înapoi: (se de*crementează, spre primul obiect al containerului)*
- Iterator bidirecţiona: (se poate şi *incrementa* şi *decrementa)*
- Iterator random: (se poate incrementa şi decrementa cu un număr oarecare de "poziţii")

Exemplu TAD **Iterator înainte**

$D_{Iter\text{-}C} = \{it_C | \text{ it – iterator peste un container C}\}$

Operaţii**:**

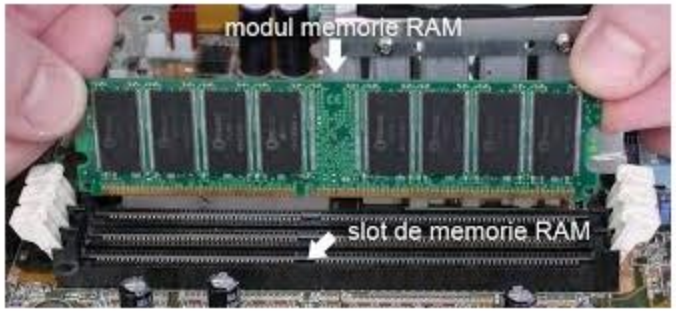
- init(ContainerSecvential C) // *Constructor, crează*
- reset() // *Setează iteratorul la pe primul obiect din C*
- next() // *Mută iteratorul spre următorul obiect*
- getCurrent() // *Returnază referinţa curentă*
- atEnd() // *Returnează* ***true*** *dacă iteratorul este la sfârşitul lui C*
- Destroy() // *Destructor, distruge iteratorul*

VECTOR

Definiţie. Un vector este un *container secvenţial* care suportă accesul direct (random) la fiecare element al sau**.**

Vectorul este cel mai simplu container şi pentru anumite aplicatii foarte eficient.

Exemplu - Memoria RAM a calculatorului



VECTOR

Clasificare:

- Static – dimensiune fixă nu pot fi adăugate/sterse elemente de memorie
- Dinamic – dimensiune modificabilă

index	**0**	**1**	**2**	**3**	**4**	**5**	**6**
valoare	0	1	4	9	16	25	36

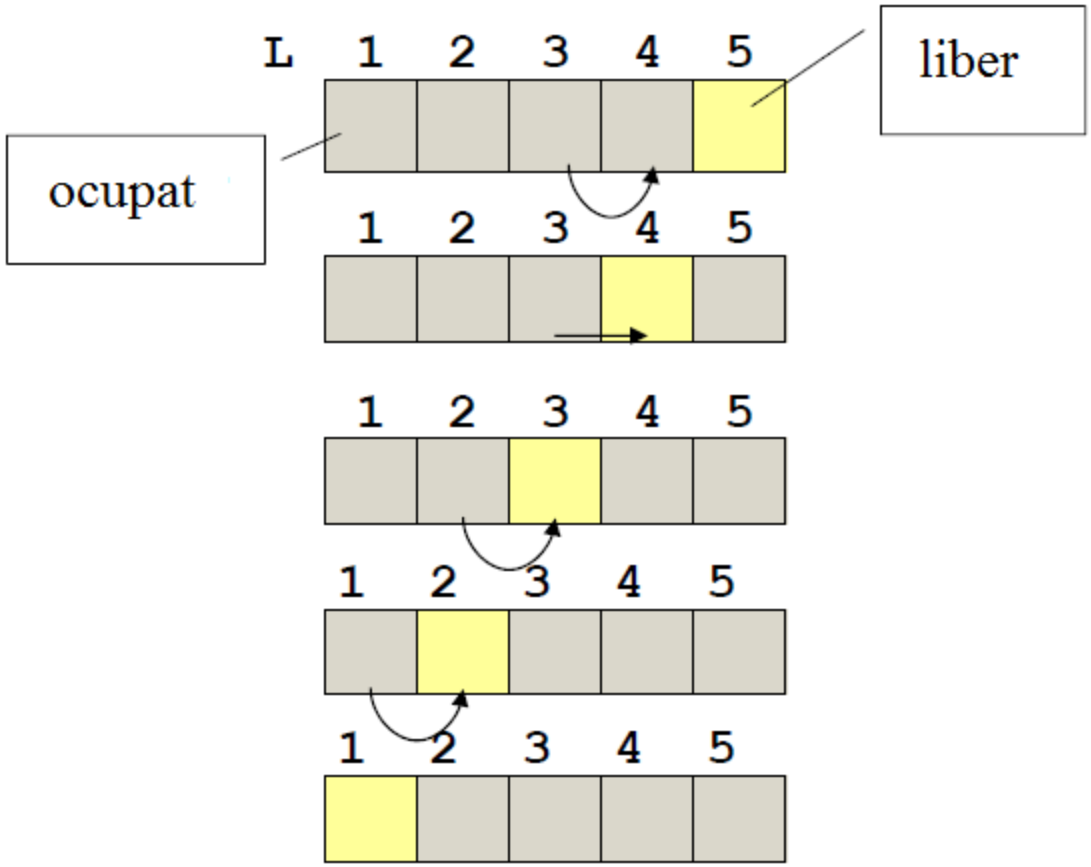
Exemplu de implementare în limbajul C:

```
int main() {
        int i;
        for (i = 0; i<7; i++ {
                v[i] = i * i
                printf („%d\n“, v[i]);
        }
        return 0;
}
```

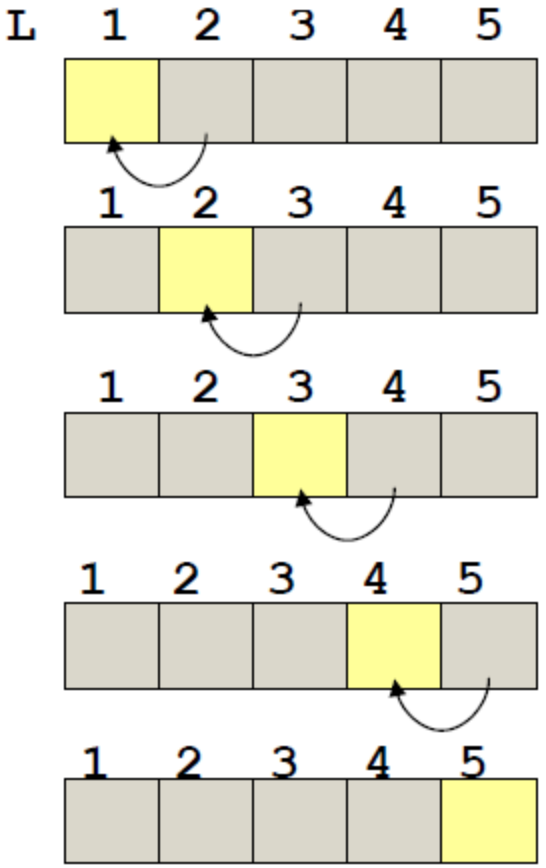
TAD Vector Static

Operaţii:	Complexitate:
• Creare	O(n)
• Set, Get *de obicei operator [] – acces aleatoriu*	O(1)
• Iniţializare (Init)	O(n)
• Dimensiune(Size)	O(1)
• Egalitate (Equal)	O(n)
• Copiere (Copy)	O(n)
• Căutare (Search)	O(n) sau $O(\log_2 n)$
• Sortare (Sort)	$O(n \log_2 n)$

Inserare element la inceputul
listei L ordonate
(altfel pierdere de informatie)



Stergere element la inceputul
listei L ordonate
(altfel pierdere de informatie)



Avantaje:

- Acces direct la fiecare element
- Accesul se face în timp constant
- Vectorii pot fi parcurşi uşor secvenţial

Dezavantaje:

- Sunt structuri statice
- În anumite cazuri nu se utilizează optim memoria
- Adăugarea unui element – operatie laborioasă
  1. Crearea unui nou vector de dimensiune mai mare
  2. Copierea elementelor în noul câmp
- Modificarea poziţiilor elementelor (de ex. într-un vector sortat) operaţie foarte laborioasă

citatepedia.info

Nu exista cunoastere innascuta, pentru motivul ca nu exista copac care sa iasa din pamant cu frunze si fructe.

Voltaire