# Liste

- In limbajul uzual cuvântul "listă" referă o "înşirare, într-o anumită *ordine*, a unor nume de persoane sau de obiecte, a unor date etc."

- Exemple de liste sunt multiple: listă de cumpărături, listă de preţuri, listă de studenţi, etc.

    - *Ordinea* în listă poate fi interpretată

        - ca un fel de „legătură" între elementele listei: după prima cumpărătură urmează a doua cumpărătură, după a doua cumpărătură urmează a treia cumpărătură, etc)
          sau
        - poate fi văzută ca fiind dată de numărul de ordine al elementului în listă (1-a cumpărătură, a 2-a cumpărătură, etc).

    - Tipul de date `Listă` care va fi definit în continuare permite implementarea în aplicaţii a acestor situaţii din lumea reală.

- O ***listă*** o putem vedea ca pe o secvenţă de elemente $< l_1, l_2, .., l_n >$ de un acelaşi tip (`TElement`), fiecare element având o *poziţie* bine determinată în cadrul listei, existând o ordine între pozițiile elementelor în cadrul listei

    - Lista poate fi văzută ca o colecţie dinamică de elemente în care este esenţială ordinea elementelor.
    - Numărul *n* de elemente din listă se numeşte *lungimea* listei.
    - O listă de lungime 0 se va numi lista *vidă*.
    - Caracterul de dinamicitate al listei este dat de faptul că lista îşi poate modifica în timp lungimea prin adăugări şi ştergeri de elemente în/din listă.

- *Poziţia* elementelor în cadrul listei este esenţială, astfel accesul, ştergerea şi adăugarea se pot face pe orice *poziţie* în listă.

- **Lista liniară**
    - o structură care fie este vidă (nu are nici un element), fie
        - are un unic prim element;
        - are un unic ultim element;
        - fiecare element din listă (cu excepţia ultimului element) are un singur succesor;
        - fiecare element din listă (cu excepţia primului element) are un singur predecesor.
    - Într-o listă liniară se pot insera elemente, şterge elemente, se poate determina succesorul (predecesorul) unui element aflat pe o anumită *poziţie*, se poate accesa un element pe baza *poziţiei* sale în listă.

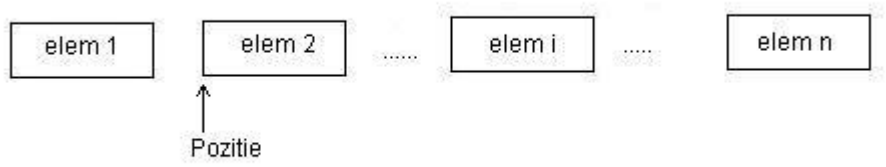
- Fiecare element al unei listei liniare are o *poziţie* bine determinată în cadrul listei.
  - este importantă prima *poziţie* în cadrul listei
  - *poziţia* unui element este relativă la listă
  - *poziţia* unui element din listă
    - identifică elementul din listă
    - poziţia elementului predecesor şi poziţia elementului succesor în listă (dacă acestea există)
  - ordine între poziţiile elementelor în cadrul listei.

***Poziţia*** unui element în cadrul listei poate fi văzută în diferite moduri:

1. ca fiind dată de ***rangul*** (numărul de ordine al) elementului în cadrul listei.
   - similitudine cu tablourile, *poziţia* unui element în listă fiind *indexul* acestuia în cadrul listei.
   - Într-o astfel de abordare, lista este văzută ca un tablou dinamic în care se pot accesa/adăuga/şterge elemente pe orice poziţie în listă.

2. ca fiind dată de o ***referinţă*** la locaţia unde se stochează elementul listei (ex: pointer spre locaţia unde se memorează elementul).

Pentru a asigura generalitatea, vom abstractiza noţiunea de ***poziţie*** a unui element în listă şi vom presupune că elementele listei sunt accesate prin intermediul unei ***poziţii*** generice.




- O **Poziţie** *p* într-o listă o considerăm ***validă*** dacă este poziţia unui element al listei.

  - dacă *p* ar fi un pointer spre locaţia unde se memorează un element al listei, atunci *p* este ***valid*** dacă este diferit de pointerul nul sau de orice altă adresă care nu reprezintă adresa de memorare a unui element al listei.

  - dacă *p* ar fi rangul (numărul de ordine al) elementului în listă, atunci *p* este ***valid*** dacă e cuprins între 1 și numărul de elemente din listă.

# TAD Lista (LIST)

**Observaţii:**

1. Tipul (abstract) de date $TPozitie$ abstractizează noţiunea de poziţie a unui element în listă (pentru a se asigura generalitatea).
2. O poziţie $p \in TPozitie$ din lista $l$ o numim *poziţie validă* dacă este poziţia unui element din lista $l$.
3. În domeniului de valori a $TPozitie$ este o valoare specială denumită poziţie nedefinită şi notată cu $\perp$. Poziţia nedefinită $\perp$ nu este o poziţie *validă* (conform celor menţionate anterior).
4. Lista vidă o notăm cu $\Phi$.

Tipul Abstract de Date LISTA:

**domeniu:**

$$\mathcal{L} = \{l \mid l \text{ este o listă cu elemente de tip } TElement, \text{ fiecare element având o poziţie unică în } l \text{ de tip } TPozitie\}$$

**operaţii:**

- creeaza($l$)
  {creează o listă vidă}
  $pre:$ $true$
  $post:$ $l \in L, l = \Phi$
- prim($l$)
  $pre:$ $l \in L$
  $post:$ $prim = p \in TPozitie$,
  $$p = \begin{cases} \text{poziția primului element din lista } l, & \text{dacă } l \neq \Phi \\ \perp, & \text{dacă } l = \Phi \end{cases}$$
- ultim (l)
  $pre:$ $l \in L$
  $post:$ $ultim = p \in TPozitie$,
  $$p = \begin{cases} \text{poziția ultimului element din lista } l, & \text{dacă } l \neq \Phi \\ \perp, & \text{dacă } l = \Phi \end{cases}$$
- valid($l, p$)
  $pre:$ $l \in L, p \in TPozitie$
  $$post: \quad valid = \begin{cases} true, & \text{dacă } p \text{ este o poziție a unui element din lista } l \\ false, & \text{altfel} \end{cases}$$

- următor$(l, p)$

  $pre:$ $l \in L, p \in TPozitie, p$ poziție validă

  $post:$ $urmator = q \in TPozitie,$

  $$q = \begin{cases} \text{poziția următoare poziției } p \text{ din lista } l, \\ \qquad \text{dacă } p \text{ nu e poziția ultimului element din lista } l \\ \perp, \qquad \text{dacă } p \text{ e poziția ultimului element din lista } l \end{cases}$$

  @ aruncă excepție dacă p nu e validă

- anterior$(l, p)$

  $pre:$ $l \in L, p \in TPozitie, p$ poziție validă

  $post:$ $anterior = q \in TPozitie,$

  $$q = \begin{cases} \text{poziția precedentă poziției } p \text{ din lista } l, \\ \qquad \text{dacă } p \text{ nu e poziția primului element din lista } l \\ \perp, \qquad \text{dacă } p \text{ e poziția primului element din lista } l \end{cases}$$

  @ aruncă excepție dacă p nu e validă

- element$(l, p, e)$

  $pre:$ $l \in L, p \in TPozitie,$ valid$(p)$

  $post:$ $e \in TElement, \quad e =$ elementul de pe poziția $p$ din $l$

  @ aruncă excepție dacă p nu e validă

- pozitie $(l, e)$

  $pre:$ $l \in L, e \in TElement,$

  $post:$ $pozitie = p \in TPozitie,$

  $$p = \begin{cases} \text{prima poziție a elementului } e \text{ din lista } l, & \text{dacă } e \in l \\ \perp, \qquad \text{dacă } e \notin l \end{cases}$$

- modifică$(l, p, e)$

  $pre:$ $l \in L, p \in TPozitie,$ valid$(p), e \in TElement$

  $post:$ elementul de pe poziția $p$ din $l' = e$

  @ aruncă excepție dacă p nu e validă

- adaugaInceput $(l, e)$

  $pre:$ $l \in L, e \in TElement$

  $post:$ elementul $e$ a fost adăugat la începutul listei $l$
  $(l' = e \oplus l)$

- adaugaSfarsit$(l, e)$

  $pre:$ $l \in L, e \in TElement$

  $post:$ elementul $e$ a fost adăugat la sfârșitul listei $l$
  $(l' = l \oplus e)$

- adaugaDupa$(l, p, e)$

  $pre:$ $l \in L, p \in TPozitie,$ valid$(p), e \in TElement$

  $post:$ elementul $e$ a fost inserat în lista $l$ după poziția $p$,
  pozitie$(l', e) =$urmator$(l', p)$

  @ aruncă excepție dacă p nu e validă

- adaugaInainte$(l, p, e)$
  $pre:$ $l \in L, p \in TPozitie$, valid$(p), e \in TElement$
  $post:$ elementul $e$ a fost inserat în lista $l$ înaintea poziției $p$,
  pozitie$(l', e) =$anterior$(l', p)$

  @ aruncă excepție dacă p nu e validă

- sterge $(l, p, e)$
  $pre:$ $l \in L, p \in TPozitie$, valid$(p)$
  $post:$ $e \in TElement$, elementul $e$ de pe poziția $p$ a fost șters din $l$

  @ aruncă excepție dacă p nu e validă

- cauta $(l, e)$
  $pre:$ $l \in L, e \in TElement$
  $post:$ $cauta = \begin{cases} adevarat, & \text{dacă } e \text{ a fost găsit în lista } l \\ fals, & \text{altfel} \end{cases}$

- vida $(l)$
  $pre:$ $l \in L$
  $post:$ $vida = \begin{cases} true, & \text{dacă } l = \Phi \\ false, & \text{dacă } l \neq \Phi \end{cases}$

- dim$(l)$
  $pre:$ $l \in L$
  $post:$ $dim = n \in Natural$,
  $n =$ numărul de elemente ale listei $l$

- distruge$(l)$
  {destructor}
  $pre:$ $l \in L$
  $post:$ $l$ a fost 'distrusa' (spațiul de memorie alocat a fost eliberat)

- iterator$(l, i)$
  $pre:$ $l \in L$
  $post:$ $i \in \mathcal{I}, i$ este un iterator pe lista $l$

Există anumite dezavantaje induse de folosirea unui parametru de tip $TPozitie$ în interfaţa listei:

1. Tipurile de referinţe concrete folosite diferă în funcţie de reprezentarea listei.
2. Interfaţa listei este destul de greoaie şi nesigură prin faptul că expune în exterior poziţiile (referinţele la locaţiile din listă).

Soluţii :

1. **STL** - *poziţia* să fie dată de un *iterator* pe listă $\Rightarrow TPozitie = Iterator$.
   - se simplifică interfaţa: operaţiile *următor*, *anterior*, *valid* şi *element* sunt operaţiile pe *iterator*.
   - *lista* (ca şi *vector*) sunt văzute ca şi containere de tip *secvenţă*: elementele sunt aranjate într-o ordine (liniară) strictă.

- reprezentarea *înlănţuită*.

2. **Java** - *poziţia* - indice (acces prin indici).

   - Accesul la elemente se face pe baza rangului, dar se permit inserări şi ştergeri la orice poziţie ($TPozitie = Intreg$, reprezintă indicele în cadrul listei).
   - O poziţie $i$ în cadrul listei $l$ este *validă* dacă $1 \leq i \leq lungime(l)$.
   - Numărul de operaţii din interfaţa listei indexate este mai mic.

Tipul Abstract de Date Lista

- cu poziţie indice - **(indexată)**

**domeniu:**

$$L = \{l \mid l = [e_1, e_2, ..., e_n], e_i \in TElement\ \forall i = 1, 2, \ldots, n\}$$

**operaţii:**

- creeaza (l)
  $pre:$ $true$
  $post:$ $l \in L, l = \Phi$ lista vidă

- adaugaSfarsit$(l, e)$
  $pre:$ $l \in L, e \in TElement$
  $post:$ elementul $e$ a fost adăugat la sfârşitul listei $l$
  $(l' = l \oplus e)$

- adauga $(l, i, e)$
  $pre:$ $l \in L, e \in TElement, i \in Intreg,$
  $i$ poziţie validă în $l$ $\vee$ $i =$ lungime$(l) + 1$
  $post:$ $l' = (e_1, \ldots, e_{i-1}, e, e_i, e_{i+1}, \cdots, e_n)$
  (pozitie$(l', e) = i$)

  @ aruncă excepţie dacă i nu e valid

- sterge $(l, i, e)$
  $pre:$ $l \in L, l = (e_1, \ldots, e_{i-1}, e_i, e_{i+1}, \cdots, e_n), i \in Intreg, i$ poziţie validă
  $post:$ $e \in TElement, e =$ elementul de pe poziţia $i$ din $l$
  $l' = (e_1, \ldots, e_{i-1}, e_{i+1}, \cdots, e_n)$
  (pozitie$(l', e) = i$)

  @ aruncă excepţie dacă i nu e valid

- cauta $(l, e)$
  $pre:$ $l \in L, e \in TElement$
  $post:$ $cauta = \begin{cases} i, & \text{dacă } i \text{ e prima pozitie pe care } e \text{ a fost găsit în lista } l \\ -1, & e \notin L \end{cases}$

- element $(l, i, e)$
  $pre:$ $l \in L, i \in Intreg, i$ poziţie validă
  $post:$ $e \in TElement, e =$ elementul de pe poziţia $i$din $l$

  @ aruncă excepţie dacă i nu e valid

- modifica $(l, i, e)$

  $pre:$ $l \in L, i \in Intreg, i$ poziție validă, $e \in TElement$

  $post:$ elementul de pe poziția $i$ din $l' = e$

  @ aruncă excepție dacă i nu e valid

- vida $(l)$

  $pre:$ $l \in L$

  $post:$ $vida = \begin{cases} true, & \text{dacă } l = \Phi \\ false, & \text{altfel} \end{cases}$

- dim $(l)$

  $pre:$ $l \in L$

  $post:$ $dim = n \in Intreg$,
  $n =$ numărul de elemente din lista $l$

- iterator$(l, i)$

  $pre:$ $l \in L$

  $post:$ $i \in \mathcal{I}, i$ este un iterator pe lista $l$

- distruge$(l)$

  $pre:$ $l \in L$

  $post:$ $l$ a fost 'distrusa' (spațiul de memorie alocat a fost eliberat)

Tipul Abstract de Date Lista

- cu poziție dată de un **iterator** -

**Operaţii din interfaţă:**

- creeaza ($l$ : Listă)
- vida ($l$ : Listă)
- dim ($l$ : Listă)
- IteratorListă prim($l$ :Listă)
- TElement element($l$ :Listă, $poz$:IteratorListă)
- TElement modifica($l$ :Listă, $poz$:IteratorListă, $e$:TElement)
- adaugaSfarsit($l$ :Listă, $e$:TElement)
- adauga ($l$ :Listă, $poz$:IteratorListă, $e$:TElement)
- TElement sterge($l$ :Listă, $poz$:IteratorListă)
- IteratorListă cauta($l$ :Listă, $e$:TElement)
- distruge ($l$ : Listă)

# Modalităţi de implementare a unei liste

- memorând elementele sale **secvenţial** într-un tablou/vector (dinamic)
  - accesul la elementele listei este *direct*
- memorând elementele sale **înlănţuit** într-o listă înlănţuită
  - accesul la elementele listei este *secvenţial*