Lista ordonată (sortată)
SORTED LIST

- Se poate impune o *ordine* între elementele unei liste sub foma unei relaţii de ordine între elementele acesteia.
  - elementele din listă sunt de **TComparabil**
- Din perspectiva unei ierarhii de containere
  - **Lista ordonată** este o un **Container ordonat/sortat**
  - **Lista ordonată** este o **Listă**
- Interfaţa tipului abstract de date **Lista ordonată**
  - modifică interfaţa **TAD Lista** astfel:
    - constuctorul va primi ca parametru relaţia de ordine între elemente
      - creeaza$(lo, \mathcal{R})$
        {creează o listă ordonată vidă}
        pre : $\mathcal{R}$ e o relaţie de ordine definită pe $TElement$ x $TElement$
        post : lo e lista vidă, relaţia de ordine devine $\mathcal{R}$
    - operaţiile de adăugare din interfaţa **TAD Lista** (*adaugaSfarsit, adaugaInceput, adaugaInainte, adaugaDupa*) se înlocuiesc cu o singură operaţie de **adăugare** (numită şi *inserare*)
      - adaugă un element în listă atfel încât să se păstreze relaţia de ordine dintre elementele listei.

        adauga(lo, e)
        {adaugă un element în listă ordonată a.î să se păstreze relaţia de ordine între elemente}
        pre : lo e o listă ordonată , $e \in TElement$
        post : e e inserat în lo, lo' rămâne ordonată
    - operaţia *modifică* (setarea unui element pe o anumită poziţie în listă) este eliminată
      - prin modificarea unui element de pe o anumită poziţie nu se poate asigura faptul că lista va rămâne ordonată.
  - pe lângă operaţiile din interfaţa minimală a Listei, putem adăuga şi alte operaţii (moştenite de la containerul **Colecţie**), spre exemplu:

    sterge (l, e)
    pre : $l \in L, e \in TElement$
    post : prima apariţie a elementului e a fost ştearsă din l

- ca şi la TAD-ul **Lista**, tipul $TPozitie$ expus în interfaţă poate fi particularizat (i.e. un indice sau un iterator), rezultând astfel
  - **Lista ordonată cu poziţie indice (indexată)** - *poziţia* este văzută ca un indice $\Rightarrow TPozitie = Intreg$.
  - **Lista ordonată cu poziţie iterator** - *poziţia* este dată de un *iterator* pe listă $\Rightarrow TPozitie = Iterator$.

Modalităţi de implementare a unei liste ordonate

Sunt aceleaşi modlităţi de implementare ca pentru o listă neordonată

- memorând elementele sale **secvenţial** într-un vector (dinamic)
  - vectorul se va memora ordonat/sortat în raport cu relaţia de ordine
    - 9 5 3 2 dacă relaţia de ordine $\mathcal{R} = \geq$
    - 2 3 5 9 dacă relaţia de ordine $\mathcal{R} = \leq$
  - operaţia **cauta** (căutarea unui element) se realizează în $O(log_2 n)$ (folosind căutare binară)
  - operaţia **adauga** va avea complexitatea timp $O(n)$
- memorând elementele sale **înlănţuit** într-o listă înlănţuită
  - lista înlănţuită va fi ordonată - memorează elementele în ordine în raport cu relaţia de ordine (a se consulta **Cursul 4**).
  - lista înlănţuită poate fi
    - simplu înlănţuită (LSI)
    - dublu înlănţuită (LDI)

Exemplu

Considerăm reprezentarea Listei ordonate cu poziţie iterator, folosind o LDI alocată dinamic.

- elementele sunt de tip **TComparabil** (**TElement**=**TComparabil**)
- ordinea între elementele listei o vom memora sub forma unei relaţii de ordine $\mathcal{R} \subseteq$ **TComparabil** x **TComparabil**, al cărei tip îl vom nota **Relaţie**. Reamintim faptul că relaţia va fi implementată în C++ sub forma unui pointer spre o funcţie (a se consulta **Cursul 4**, pentru detalii).

Reprezentarea listei şi a iteratorului pe listă sunt date mai jos

Nod
e: TElement *//infomaţia utilă nodului*
urm: ↑ Nod *//adresa la care e memorat următorul nod*

prec: ↑ Nod *//adresa la care e memorat nodul anterior*

Lista
prim: ↑ Nod*//adresa primului nod din listă*
ultim: ↑ Nod*//adresa ultimului nod din listă*
$\mathcal{R}$: **Relaţie***//relaţia de ordine între elemente*

IteratorLista
l: Lista*//referinţă către listă*
curent:↑ Nod*//adresa nodului curent din listă*

Descriem mai jos, în Pseudocod, operaţiile **creeaza** (constructorul listei ordonate) şi **cauta** (localizarea unui element în listă). Reamintim specificaţia acestei operaţii

cauta (lo, e)
pre : lo listă ordonată în raport cu o relaţie de ordine $\mathcal{R}, e \in TElement$
post : returnează un iterator: prima poziţie pe care apare elementul sau iterator invalid

```
Subalgoritm creeaza(lo, rel)
  {pre: rel: Relatie}
  {post: lo e ListaOrdonata, lo e lista vidă, ordinea între elementele listei
  e rel }
  {lista e vidă}
  lo.prim ← NIL
  lo.ultim ← NIL
  {setăm relaţia}
  lo.R ← rel
SfSubalgoritm
```

- Complexitate: $\theta(1)$

La căutarea primei poziţii pe care apare un element în lista ordonată, trebuie să ţinem cont de faptul că lista e ordonată. De exemplu, dacă lista e 4 7 9 11 ($\mathcal{R} = \leq$)

1. dacă vrem să căutăm **3** ($3 < 4$) sau **12** ($12 > 11$), suntem siguri de eşuarea căutării.

2. dacă vrem să căutăm elementul **e=6**, în momentul în care ajungem la **7** nu are rost să continuăm căutarea - am găsit un element mai mare decât **e** (în cazul general, am găsit un element care nu e în relaţia $\mathcal{R}$ cu **e**).

```
Functia cauta(lo, e)
  {pre: lo: ListaOrdonata}
  {post: returnează un iterator: prima poziţie pe care apare elementul sau
  iterator invalid}
  {se creează un iterator invalid pe lista lo}
  {apelăm constructorul iteratorului}
  creeaza(it, lo)
```

```
    {iteratorul e invalid}
    it.curent ← NIL
    {dacă e sigur că e nu apare în listă (caz 1 de mai sus)}
    Daca (lo.prim = NIL) ∨ (¬ ([lo.prim].e  ℛ e)) ∨ (¬ (e ℛ  [lo.ultim].e))
    atunci
      {returnăm iterator invalid)}
      cauta ← it
    altfel
      {căutăm până găsim elementul, sau e sigur că nu apare în listă (cazul 2
      de mai sus)}
      p ← lo.prim
      CatTimp (p ≠ NIL) ∧ ([p].e ≠ e) ∧ (¬ (e ℛ  [p].e)) executa
        p ← [p].urm
      SfCatTimp
      {dacă am găsit elementul, setăm iteratorul pe nodul găsit)}
      Daca (p ≠ NIL) ∧ ([p].e = e) atunci
        it.curent ← p
      SfDaca
      {returnăm iteratorul)}
      cauta ← it
    SfDaca
  SfFunctia
```

- Complexitate: $O(n)$, n fiind numărul de elemente din listă

În directorul TAD Lista Ordonata (**Curs 5**) găsiţi implementarea parţialăîn limbajul C++ a containerului **Lista ordonată** cu poziţie iterator (reprezentarea este sub forma unei LDI, folosind alocare dinamică pentru reprezentarea înlănţuirilor). Atenţie: operaţia de adăugare nu e completă (e tratat doar cazul în care elementul trebuie adăugat la începutul listei).