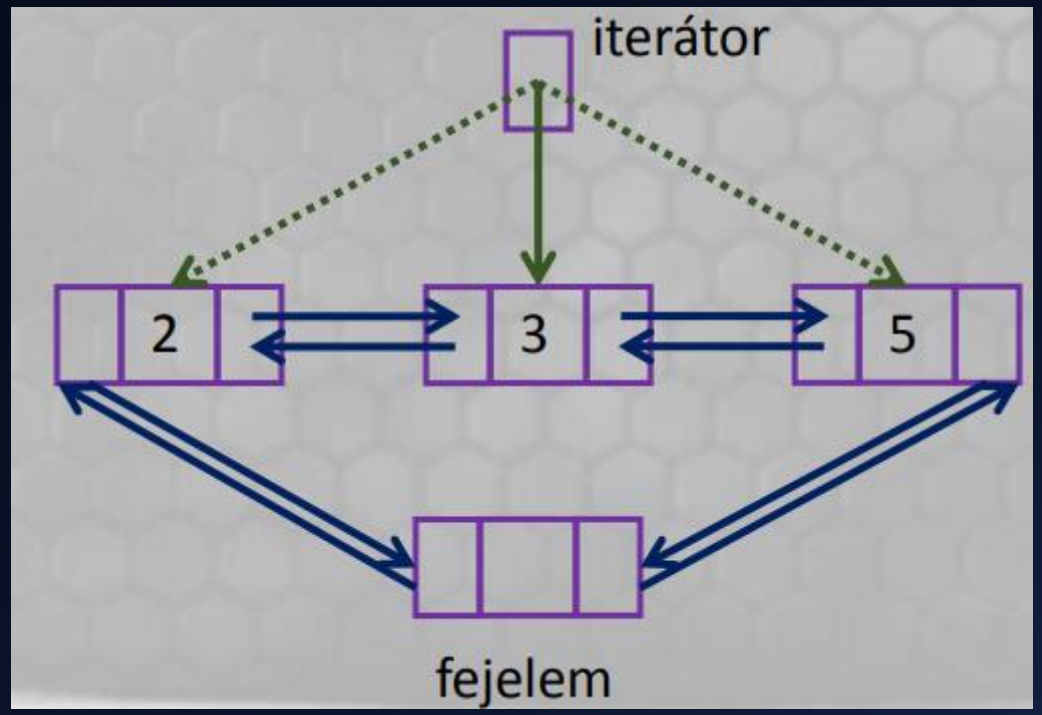
Objektum orientált programozás C++ nyelven 9. STL tárolók

PEKÁRDY MILÁN – PANNON EGYETEM

PEKARDY@DCS.UNI-PANNON.HU

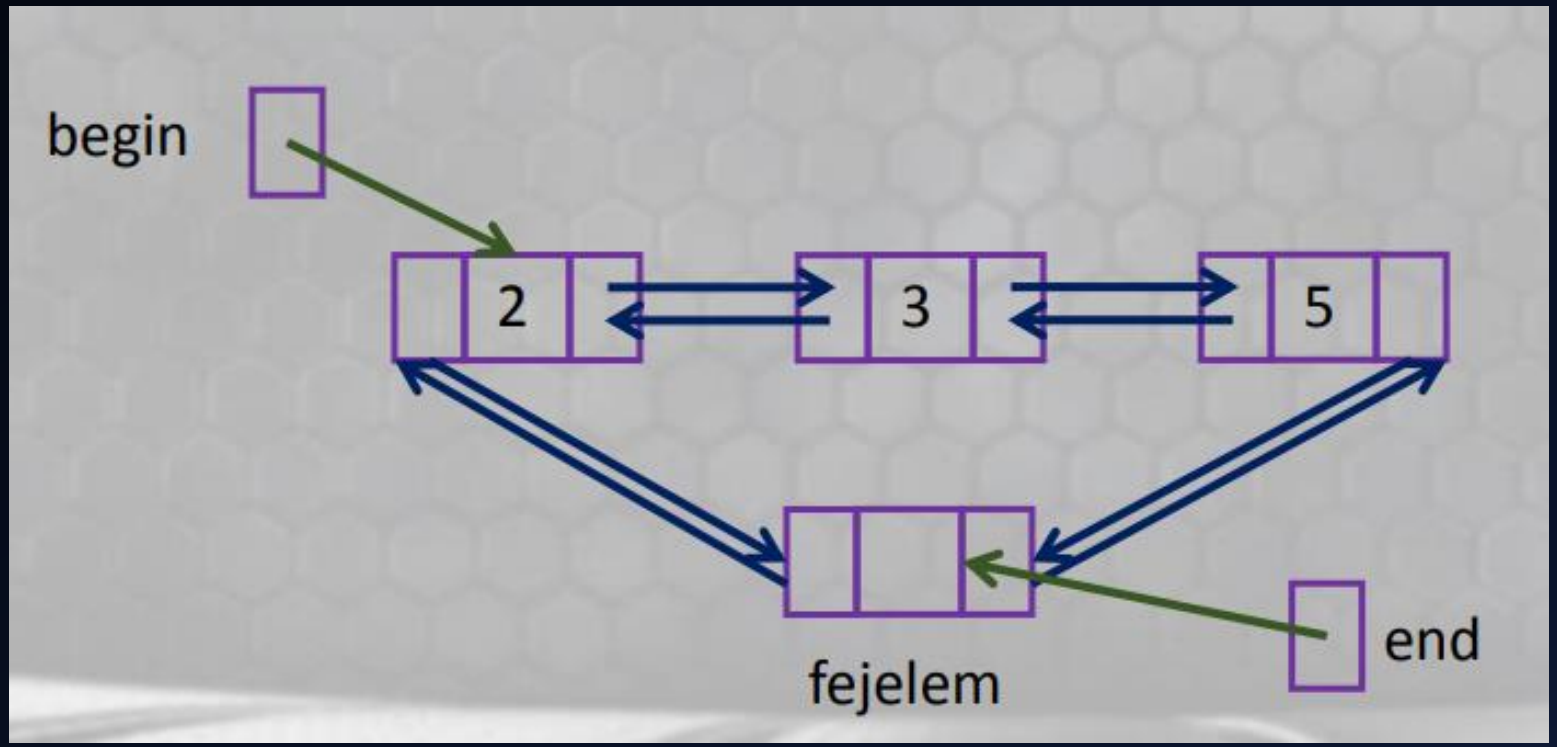
Iterátorok - bevezetés

- A lista által nyújtott interfész a tárolt elemek elérésére
- Neve onnan jön, hogy jellemzően ciklusváltozóként használjuk
- Hasonló, mint tömbök esetén az index
- Eltárolja a lista egy elemének címét
- Képes előre-hátra lépni a listában



Iterátorok – alap iterátor pozíciók

- Begin: az első elemre mutat
- End: az utolsó utáni elemre mutat (körkörös lista esetén a fejelemre)



STL – Standard Template Library

- Nem a C++ nyelv része, hanem egy kiegészítés
- Annyire elterjedt, hogy minden C++ fordító alapértelmezetten tartalmazza
- Főbb részei:
  - Tárolók
  - Iterátorok
  - Algoritmusok

STL tárolók

- Sorrendi tárolók
  - Az adatok tárolása egy szekvenciális szerkezettel történik, amelynek egyértelműen van eleje és vége
- Tároló adapterek
  - A tároló osztály belső adattagként egy másik tároló osztályt használ, és speciális elérési lehetőségeket (interfészt) biztosít.
- Asszociatív tárolók
  - Az elemek tárolása nem soros módon történik, az elemeket nem a sorban elfoglalt helyük adja meg, hanem egy másik adat
- Egyéb tárolók

STL tárolók

- Sorrendi tárolók:
  - vector, deque, list
- Tároló adapterek:
  - stack, queue, priority_queue
- Asszociatív tárolók:
  - set, multiset, map, multimap
- Egyéb tárolók
  - bitset, valarray
- A C++11 ezt a listát kiegészíti újabb tárolókkal

list

- Kétirányban láncolt listán alapul
- Az elemek tárolása szekvenciális
- Minden elemre meg lehet mondani, hogy mi van előtte és utána
- Tetszőleges helyen lévő elem elérhető iterátorral

list – alap függvények

`iterator begin();`	Visszatér egy iterátorral, ami az első elemre mutat
`iterator end();`	Visszatér egy iterátorral, ami az utolsó utáni elemre mutat
`reverse_iterator rbegin();`	Visszatér egy fordított irányú iterátorral, ami az utolsó elemre mutat
`reverse_iterator rend();`	Visszatér egy fordított irányú iterátorral, ami az első előtti elemre mutat
`bool empty() const;`	Visszaadja, hogy üres-s a lista
`size_type size() const;`	Visszaadja a listában található elemek számát

- A size_type egy típusdefiníció, jellemzően unsigned int

list – alap függvények

`reference front();`	Visszatér a lista első elemének referenciájával
`reference back();`	Visszatér a lista utolsó elemének referenciájával
`void push_front (const value_type& val);`	Beszúrja a paramétert a lista elejére
`void push_back (const value_type& val);`	Beszúrja a paramétert a lista végére
`void pop_front();`	Kitörli a lista első elemét
`void pop_back();`	Kitörli a lista utolsó elemét

- A value_type a template paraméter a listához, vagyis a tárolt adat típusa.
- A reference egy típusdefiníció, jellemzően value_type&

list – alap függvények

`iterator insert (iterator position, const value_type& val);`	Beszúr a listába a megadott pozícióra 1 vagy több példányt ugyanabból az értékből (1. és 2. verzió), vagy más iterátorokkal megadott tartományt (3. verzió).
`void insert (iterator position, size_type n, const value_type& val);`	
`template <class InputIterator> void insert (iterator position, InputIterator first, InputIterator last);`	
`iterator erase (iterator position);`	Kitörli a listából a megadott iterátor (1.) vagy iterátorok (2.) által jelzett értékeket.
`iterator erase (iterator first, iterator last);`	
`void clear();`	Kitöröl mindent a listából.

list – kiegészítő függvények

`void remove (const value_type& val);`	Érték szerint töröl, minden előfordulást, destruktort meghívja.
`void sort();`	< operátort használva rendezi a lista elemeit.
`template <class Compare>` `  void sort (Compare comp);`	Saját összehasonlító függvénnyel rendezi a listát.
`void unique();`	Rendezett listából törli a duplikátumokat, a legelsőt hagyja meg.

vector

- Dinamikusan átméreteződő tömbön alapul
- Az elemek tárolása szekvenciális
- Tetszőleges helyen lévő elem elérhető, akár iterátorral, akár indexeléssel
- Ha megtelik a tömb 2-szer nagyobb tömböt foglal utána.
- Mivel átméreteződik, az iterátorok használata csak rövid távra ajánlott

vector – alap függvények

`iterator begin();`	Visszatér egy iterátorral, ami az első elemre mutat
`iterator end();`	Visszatér egy iterátorral, ami az utolsó utáni elemre mutat
`reverse_iterator rbegin();`	Visszatér egy fordított irányú iterátorral, ami az utolsó elemre mutat
`reverse_iterator rend();`	Visszatér egy fordított irányú iterátorral, ami az első előtti elemre mutat
`bool empty() const;`	Visszaadja, hogy üres-s a vektor
`size_type size() const;`	Visszaadja a vektorban található elemek számát

- A size_type egy típusdefiníció, jellemzően unsigned int

vector – alap függvények

`reference front();`	Visszatér a vektor első elemének referenciájával
`reference back();`	Visszatér a vektor utolsó elemének referenciájával
`void push_back (const value_type& val);`	Beszúrja a paramétert a vektor végére
`void pop_back();`	Kitörli a vektor utolsó elemét
`reference at (size_type n);`	Visszaad egy referenciát az adott helyen található elemre
`reference operator[] (size_type n);`	

- A vector osztálynak nincs pop_front és push_front függvénye, mivel egy átméreteződő tömbnek a végéhez szokás hozzáfűzni.
- A value_type a template paraméter a vektorhoz, vagyis a tárolt adat típusa.
- A reference egy típusdefiníció, jellemzően value_type&.

vector – alap függvények

`explicit vector (const allocator_type& alloc = allocator_type());`	Fontosabb konstruktorok. Az `allocator_type`-al nem kell foglalkozni. Az első az alapértelmezett, a második pedig a megadott mérettel hozza létre a vektort, és feltölti a második paraméterben megadott értékkel.
`explicit vector (size_type n, const value_type& val = value_type(), const allocator_type& alloc = allocator_type());`	
`void resize (size_type n, value_type val = value_type());`	Átméretezi a vektor, és növelés esetén az új elemeket a megadott paraméterrel tölti fel.

- A value_type a template paraméter a vektorhoz, vagyis a tárolt adat típusa.
- A reference egy típusdefiníció, jellemzően value_type&

map

- Asszociatív tároló, az elemek nem szekvenciálisan követik egymást.
- Jellemzően bináris keresőfán alapul
- Minden elem két részből áll
- Kulcs, ami azonosítja, ez általában egy rendezhető adattípus
- Adat, ami a tényleges adatot tárolja
- Nem tartalmaz ismétlődő kulcsokat
- Egy int-eket tároló tömb esetén:

0	1	2	3	4	Index
12	24	1	0	11	Tárolt érték

- Asszociatív tömb esetén:

Alan	Hugh	Josh	Andrew	Kate	Kulcs
12	24	1	0	11	Érték

map – kiegészítő adatszerkezet: pair

- A pair egy sablon adatszerkezet, amely két adatot fog össze
- Struktúra két, publikus adattagal: first és second

```
template <class T1, class T2> struct pair;
// Példa:
pair<int, double> id_pair(3,4.5);
cout << id_pair.first << ", " << id_pair.second << endl; // 3, 4.5
```

map – alap függvények

`iterator begin();`	Visszatér egy iterátorral, ami az első elemre mutat
`iterator end();`	Visszatér egy iterátorral, ami az utolsó utáni elemre mutat
`reverse_iterator rbegin();`	Visszatér egy fordított irányú iterátorral, ami az utolsó elemre mutat
`reverse_iterator rend();`	Visszatér egy fordított irányú iterátorral, ami az első előtti elemre mutat
`bool empty() const;`	Visszaadja, hogy üres-s a map
`size_type size() const;`	Visszaadja a map-ben található elemek számát

- A size_type egy típusdefiníció, jellemzően unsigned int.
- A mapped_type a második sablon paraméter, ami a tárolt adat típusát azonosítja.
- A key_type az első sablon paraméter, ami a kulcs típusát azonosítja.
- A value_type egy típusdefiníció, ami egy teljes elemet jelöl: pair

map – alap függvények

`mapped_type& operator[] (const key_type& k);`	Visszaad egy referenciát az adott helyen található elemre. Ha nincs elem a megadott kulccsal, akkor létrehozza, és azután adja vissza. A kulcshoz tartozó értéket frissíti.
`pair<iterator,bool> insert (const value_type& val);`	Beszúrja a megadott elemet, ha a kulcs még nem létezik. Ha a kulcs már létezik, akkor nem frissít. Visszatér a létező elem iterátorával.
`void erase (iterator position);`	Kitörli az iterátorral (1.) vagy értékkel (2.) megadott elemt a map-ből
`size_type erase (const key_type& k);`	
`void clear();`	Kiüríti a teljes tárolót
`iterator find (const key_type& k)`	Kulcs alapján megkeres egy elemet a map-ben, és visszaadja az iterátorát. Ha nincs ilyen, akkor map::end-el tér vissza