Pázmány Péter Katolikus Egyetem
Információs Technológiai és Bionikai Kar

Bevezetés a Programozásba II

12. előadás

Adatszerkezetek alkalmazása (Standard Template Library)

© 2014.05.19. Giachetta Roberto
groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

Adatszerkezetek alkalmazása
Adatszerkezetek

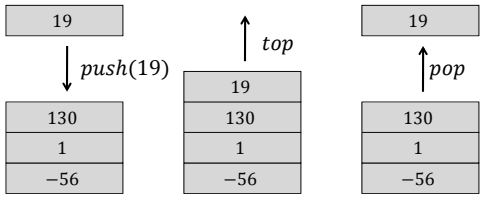
- *Adatszerkezet*nek nevezzük adott típusú adatok valamilyen megadott struktúrában felépített halmazát
- A főbb adatszerkezetek a legtöbb programozási nyelvben definiáltak, ezeket *gyűjtemények*nek (*collection*) nevezzük
  - *tömb* (vektor): elemek rögzített hosszú sorozata, amelyben bármely elem közvetlenül elérhető
  - *verem* (LIFO): a végén bővíthető, az utolsó elem érhető el
  - *sor* (FIFO): a végén bővíthető, az első elem érhető elem
  - *lista*: tetszőleges helyen bővíthető
  - *asszociatív tömb*: speciális értékekkel indexelt, tetszőleges helyen bővíthető

Adatszerkezetek alkalmazása
A Standard Template Library

- A *Standard Template Library* (*STL*) egy olyan programkönyvtár, amely előre megírt sablonos adatszerkezeteket és algoritmusokat tartalmaz
  - minden típusra hatékonyan alkalmazhatóak
  - szabványosan, objektumorientáltan implementáltak
  - robosztusak (az adatszerkezetek gyorsan cserélhetőek)
- Az STL-en keresztül adatainkat különféle adatszerkezetekbe szervezhetjük, és rajtuk sokféle beépített algoritmust futtathatunk
  - pl. keresések, rendezések, kiválasztások,…

Adatszerkezetek alkalmazása
A verem

- A *verem* (*stack*), más néven LIFO (Last-In-First-Out) egy olyan adatszerkezet, amelynek csak a tetejére tudunk helyezni új elemet (*push*), illetve csak a tetőelemet kérdezhetjük el (*top*), és vehetjük ki (*pop*), tehát csak azt az elemet tudjuk elérni és kivenni, amit utoljára helyeztünk bele



Adatszerkezetek alkalmazása
A verem

  - mérete tetszőlegesen növelhető (a maximális méret korlátozható), és mivel csak a tetőelem érhető el, hatékonyan implementálható
- A verem típust a `stack` osztály valósítja meg, fontosabb műveletei:
  - `push`: érték behelyezése a verem tetejére
  - `pop`: tetőelem kivétele
  - `top`: tetőelem lekérdezése
  - `size`: méret lekérdezése
  - `empty`: üres-e a verem

Adatszerkezetek alkalmazása
A verem

- Pl.:

```
#include <stack>
using namespace std;
…
stack<int> intStack; // üres verem létrehozása

intStack.push(-56); // elemek behelyezése
intStack.push(1);
intStack.push(130);
intStack.push(19);

cout << intStack.top() << endl; // eredmény: 19
intStack.pop();
cout << intStack.top() << endl; // eredmény: 130
```

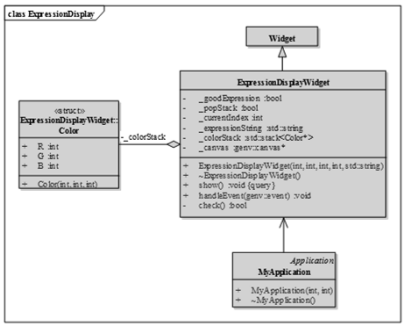
Adatszerkezetek alkalmazása
Példa

Feladat: Készítsünk egy alkalmazást, amelyben egy kifejezést színezhetünk a zárójelezése szerint.

- minden egyes új zárójel nyitása a kifejezés egy új szintje, amelyet véletlenszerűen színezünk ki
- amikor egy zárójeles rész véget ér, a korábbi színre kell visszalépnünk, ez veremmel szabályozható
- mindezt egy megjelenítő vezérlő segítségével (**`ExpressionDisplayWidget`**), amely egy veremben tárolja a véletlenszerűen generált színeket (**`_colorStack`**)
- a kifejezés helyes zárójelezettségét ellenőrizzük is (**`check`**), szintén veremmel

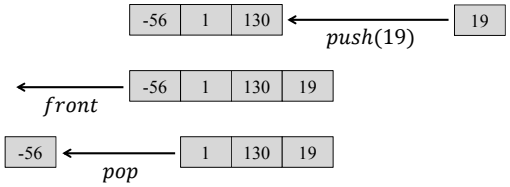
Adatszerkezetek alkalmazása
Példa

Tervezés:



Adatszerkezetek alkalmazása
A sor

- A *sor* (*queue*), más néven FIFO (First-In-First-Out) egy olyan adatszerkezet, amelynek a végére tudunk helyezni új elemet (*enqueue*, *push*), illetve csak az elejét kérdezhetjük el (*first*, *front*), és vehetjük ki (*dequeue*, *pop*), tehát csak azt az elemet érhetjük el és vehetjük ki, amit a legrégebben helyeztünk be



Adatszerkezetek alkalmazása
A sor

- az első elem utáni elemek rejtettek (esetleg az utolsü lekérdezhető), mérete tetszőlegesen növelhető
- A sor típust a **`queue`** osztály valósítja meg, fontosabb műveletei:
  - **`push`**: érték behelyezése a sor végére
  - **`pop`**: első elem kivétele
  - **`front`**: első elem lekérdezése
  - **`back`**: utolsó elem lekérdezése
  - **`size`**: méret lekérdezése
  - **`empty`**: üres-e a sor

Adatszerkezetek alkalmazása
A sor

- Pl.:

```
queue<int> intQueue; // üres verem létrehozása
intQueue.push(-56); // elemek behelyezése
intQueue.push(1);
intQueue.push(130);
intQueue.push(19);

cout << intQueue.front() << endl; // eredmény: -56
cout << intQueue.back() << endl; // eredmény: 19

intQueue.pop(); // elemek kivétele
intQueue.pop();
cout <<  intQueue.front() << endl;
     // eredmény: 130
```


Adatszerkezetek alkalmazása
Példa

Feladat: Készítsünk egy egyszerű sürgősségi betegellátó játékprogramot.

- a kórházba betegek érkeznek, mindegyik meggyógyítása adott időt vesz igénybe
- a gyógyítást orvosok végzik, akiket a sürgősségire külön kell behívni, és ezért külön fizetést kapnak (amíg bent vannak, és haza is lehet küldeni őket)
- a sürgősségnek adott költségvetése van, amely mindig fix összeggel bővül, és annak mindig pozitívban kell maradnia
- nem szabad, hogy a sürgősségit elárasszák a betegek, így mindig megfelelő orvosszámnak kell rendelkezésre állnia

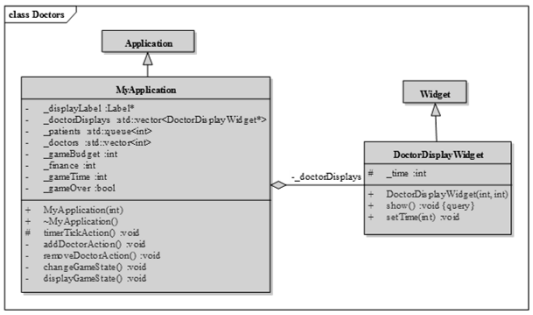
Adatszerkezetek alkalmazása
Példa

Tervezés:

- az alkalmazást grafikus felülettel látjuk el, és vezérlőkből alkotjuk meg
- információk kiírása címkét (**`Label`**), tevékenységre két gombot (**`Button`**) használunk, ezen felül az orvosok megjelenítésére új vezérlőket hozunk létre (**`DoctorDisplayWidget`**)
- a játékkal kapcsolatos információkat a saját alkalmazásban (**`MyApplication`**) helyezzük el (betegek, orvosok, játékidő, pénzmennyiség)
- a betegeket egy sor segítségével kezeljük (**`_patients`**)

Adatszerkezetek alkalmazása
Példa

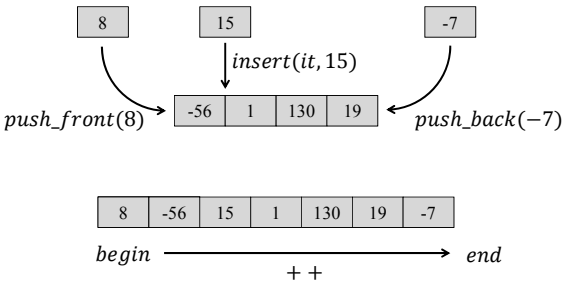
Tervezés:



Adatszerkezetek alkalmazása
A lista

- A lista (*list*) egy olyan szekvenciális adatszerkezet, amely műveletek széles skáláját biztosítja
  - bármely elemet, illetve több egymás utáni elemet módosíthatunk, törölhetünk, bármely elem elé, vagy után beszúrhatunk egy, vagy több új elemet
  - az egyes elemek elérése nem indexeléssel, hanem *bejáró* (*iterator*) segítségével történik
  - a lista egyes részeit kicserélhetjük, vagy kivehetjük másik listába, listákat összefűzhetünk egy közös listába
  - az elemek sorrendjét megfordíthatjuk, vagy rendezhetjük a listát tetszőleges módon

Adatszerkezetek alkalmazása
A lista



Adatszerkezetek alkalmazása
A lista

- Pl.:

```
list<int> intList;
intList.push_back(10); // lista: 10
intList.push_back(4); // lista: 10,4
intList.push_front(-3); // lista: -3,10,4
intList.pop_back(); // lista: -3,10
intList.reverse(); // lista: 10, -3

list<int>::iterator it; // bejáró létrehozása
for (it = intList.begin();
     it != intList.end(); it++){
   cout << *it << endl; // elemek kiírása
   *it = 0;  // elemek kinullázása
}
```


Adatszerkezetek alkalmazása
A lista

```
intList.push_back(-4); // lista: 0,0,-4
intList.unique(); // lista: 0,-4
intList.sort(); // lista: -4,0

it = intList.begin(); iter++;
intList.insert(iter, 5); // lista: -4,5,0
intList.insert(iter, 2); // lista: -4,2,5,0
it--; intList.erase(it); // lista: 2,5,0

// fordított bejárás:
for (list<int>::reverse_iterator rit =
     intList.rbegin(); rit != intList.rend();
     rit++){
   cout << *rit << endl; // elemek kiírása
}
```

Adatszerkezetek alkalmazása

Az asszociatív tömb

- Az *asszociatív tömb* (*map*) egy olyan vektor, amelynek indexelése nem csupán a rögzített módon történhet (egész számok 0-tól), hanem tetszőleges típusú, és azon belül tetszőleges elemkombinációjú
  - az indexeket *kulcs*oknak nevezzük, kulcs alapján kérdezhetünk le elemeket, illetve az alapján törölhetünk
  - az elemeknek nincs sorrendje, és csak azokat a kulcsokat érhetjük el, amelyeket már behelyeztünk
  - lekérdezhetjük az összes elemet *bejáró* (*iterátor*) segítségével, ekkor kulcs/érték pár mutatókat kapunk (a kulcs az első elem, az érték a második)

Adatszerkezetek alkalmazása

Az asszociatív tömb

- Az asszociatív tömböt a **`map`** osztály valósítja meg, műveletei:
  - **`[]:`** elem lekérdezése és írása
  - **`find:`** keresés kulcs alapján
  - **`insert`**: elem beszúrása a megadott helyre
  - **`erase`**: elem törlése a megadott helyről
  - **`swap`**: elemek kicserélése
  - **`clear`**: lista kiürítése
  - **`size`**: méret lekérdezése
- A bejáró a **`map::iterator`** típuson keresztül érhető el, és egy mutatót ad az elempárokra, ahol a kulcsot a **`first`**, az értéket a **`second`** attribútummal érhetjük el

Adatszerkezetek alkalmazása

Az asszociatív tömb

- Pl.:

```
map<string, int> strMap;
strMap["students"] = 67; // elemek felvétele
strMap["teachers"] = 5;
strMap["courses"] = 1;
if (strMap.find("courses")) // kulcs keresése
   cout << strMap["courses"] << endl; // kiírja: 1

// bejárás:
for (map<string, int>::iterator it =
     strMap.begin(); it != strMap.end(); it++)
   cout << it->first << " => " << it->second
        << endl;
   // kiírja a kulcsot, majd az értéket
```