HÁZI FELADAT

Programozás alapjai 2.

Végleges

VONATJEGY ELÁRUSÍTÓ

Kurcsi Norbert Y3ZTEI

-2022. május 16.-

TARTALOM

Feladat

Tervezze meg egy vonatjegy eladó rendszer egyszerűsített objektummodelljét, majd valósítsa azt meg! A vonatjegy a feladatban mindig jegyet és helyjegyet jelent együtt. Így egy jegyen minimum a következőket kell feltüntetni:

- vonatszám, kocsiszám, hely
- indulási állomás, indulási idő
- érkezési állomás, érkezési idő

A rendszerrel minimum a következő műveleteket kívánjuk elvégezni:

- vonatok felvétele
- jegy kiadása

A rendszer később lehet bővebb funkcionalitású (pl. késések kezelése, vonat törlése, menetrend stb.), ezért nagyon fontos, hogy jól határozza meg az objektumokat és azok felelősségét.

Valósítsa meg a jeggyel végezhető összes értelmes műveletet operátor átdefiniálással (overload), de nem kell ragaszkodni az összes operátor átdefiniálásához! A megoldáshoz **ne használjon STL tárolót**!

Feladatspecifikáció

Vonat adatbázis kiválasztása

Miután elindul a program, egy olyan menü jelenik meg, ahol a felhasználó ki tudja választani, hogy fájlból töltsön be egy már meglévő adatbázist, amelyet ezzel a programmal hoztak létre korábban, vagy kezdjen egy új adatbázist, amelyben még nincs semmiféle adat tárolva, és majd neki lesz a dolga, hogy az adatokat felvegye a rendszerbe. Ezek mellet egy kilépés opció is lesz, amit kiválasztva a program megáll.

Adatbázisok betöltése

Ha a felhasználó az “Adatbázis betöltése” opciót választja, akkor a következő nézetben a már létrehozott adatbázisok kilistázódnak neki, és lehetősége nyílik ezek közül kiválasztani a neki megfelelőt. Rendezni is lehet az adatbázisokat név, illetve módosítás dátum alapján, a könnyebb azonosítás érdekében. Miután egy adatbázis kiválasztásra kerül, a program az általános menüt jeleníti meg.

Általános menü

Az általános menüben a felhasználónak lehetősége nyílik új vonatot felvenni. Ehhez specifikálnia kell, hogy hány kocsija lesz a vonatnak és a kocsik milyen osztályúak, illetve azt is, hogy helyközi vagy intercity vonatról van-e szó. Továbbá meg kell adnia, hogy honnan indul a vonat és milyen állomásokon keresztül, hova érkezik. A helyközi vonatok esetén a vonatnak több megállója is lehet, intercity vonat esetén csak két város között közlekedhet a vonat. A programnak meg kell adni azt is, hogy osztályonként mennyibe kerül egy jegy. Helyközi vonatok esetén ez az ár az első állomástól az utolsóig vonatkozik. Meg kell adni az indulás, illetve érkezési időt, és a program ettől függően kiszámítja, hogy melyik állomásra mikor fog odaérni a szerelvény.

Miután egy vonat járatot létrehoztak, a későbbiekben a paramétereit változtatni is van lehetőség: menetrend megváltoztatása, útvonal megváltoztatása, osztályonkénti árak megváltoztatása, illetve, ha a felhasználó úgy dönt, akkor törölni is lehet egy adott vonatot.

A programban jegyeket is lehet eladni. Ehhez meg kell adni egy kiindulópontot és egy célállomást, majd a program kilistázza a megfelelő vonatokat. Jegyek törlésére is van lehetőség, ha ez megtörténik, akkor az adott hely újra szabad lesz a vonaton.

A programban egy listázás opció is van, amelynek segítségével ki lehet listázni az összes vonatjáratot, szűrni őket megállók szerint, vagy akár rendezni név szerint. Az eladott jegyeket is ki lehet listázni és rendezni az áruk szerint.

Fontos tulajdonsága a programnak, hogy bármennyi adatot fel lehet venni, nincs semmiféle korlátozása az adatok hosszára és mennyiségére.

- ### Mentés

Miután a felhasználó elvégezte a szükséges módosításokat és műveleteket, rendelkezésére áll egy olyan opció, hogy elmentse az adatokat. Választhatja azt, hogy a korábban megnyitott adatbázist módosítsa az elmentéssel, vagy egy új mentést végezzen. Ha az utóbbit választja, akkor kell adjon egy nevet is a mentésének. Mentés után újra a 2.1. pontban leírt menü jelenik meg.

- ### Navigáció a programban

A program használatához a felhasználó a billentyűzete segítségével tud majd navigálni a különböző menüpontok között, illetve az adatok beviteléhez is ezt tudja majd használni.

Pontosított feladatspecifikáció

A program indításakor egy menü jelenik meg, amelynek tartalma a következő:

- **Meglévő adatbázis betöltése:** Ezt az opciót az 1-es billentyűvel választhatjuk ki. Ekkor a program kilistázza nekünk a már létrehozott adatbázisokat. Ezeket rendezhetjük név, illetve módosítás dátuma szerint. Ezután a megfelelő név begépelésével betölthetjük az adott adatbázist. Ha a nevet a felhasználó rosszul gépeli be, vagy rossz nevet ad, akkor a program értesíteni fogja, hogy ilyen nevű adatbázis nincs lementve.

- **Utolsónak módosított adatbázis betöltése:** Ezt az opciót a 2-es billentyűvel éri el a felhasználó. Ennek segítségével gyorsan kiválasztható az elmentett adatbázisok közül az, amin utoljára dolgoztak. Ha nincs egy elmentett adatbázis sem, akkor ezt egy üzenettel jelzi a program.
- **Új adatbázis:** Ezt a menüpontot a 3-as billentyűvel érheti el a felhasználó. Ekkor egy üres adatbázis jön létre, amelyben még semmi adat sincs elmentve.
- **Kilépés:** Ezt választva le lehet állítani a programot.

Mind a három fenti almenüből tovább lépve a főmenübe lép a program, ahol különböző lekérdezéseket lehet megjeleníteni és módosításokat lehet végrehajtani az aktuális adatbázissal kapcsolatban. Ezek az opciók a következők:

1. **Vonat hozzáadása [1]:** Először a vonat típusát kell megadni majd a vonat egyedi számát. Ezután a program sorban fogja kérni, hogy hány első osztályú, másod osztályú kocsija van a szerelvénynek, illetve az osztályonkénti árat. Ezt követően meg kell adni az indulás és érkezés dátumát és időpontját. Majd ezek után intercity vonat esetén a kezdő állomást és végállomást. Helyközi vonat esetén azt kell megadni, hogy hány megállója van a szerelvénynek, és az állomások neveit egyenként.

2. **Jegyvásárlás [2]:** Ehhez meg kell adni. hogy honnan indulunk és hová szeretnénk érkezni. A program megnézi, hogy van-e megfelelő vonat és kilistázza őket. Ha nincs megfelelő vonat, akkor ezt jelzi egy üzenet formájában. A kilistázott vonatok listájában az is szerepel, hogy az adott vonat milyen típusú, mikor indul és érkezik a megadott helyiségbe, illetve, hogy mennyibe kerül egy jegy első és másodosztályon. A felhasználónak meg kell adni a vonat számát és a kocsi osztályát. Ezek után a program egy megerősítést ír, amelyben fel lesz tüntetve a kocsi száma, illetve a lefoglalt hely.

3. **Vonat törlése [3]:** Kilistázódnak a vonatok, majd meg kell adni, hogy melyik vonatot szeretnénk törölni. A vonat törlésével az adott vonatra vonatkozó jegyek is törlődnek.

4. **Jegy törlése [4]:** Kilistázódnak a jegyek, ezután a megfelelő sorszámot megadva törölni lehet a jegyet. Ekkor az adott hely újra üres lesz a vonaton.

5. **Útvonal megváltoztatása [5]:** Intercity vonat esetén meg kell adni az új indulás és érkezési állomást. Helyközi vonat esetén meg kell adni, hogy hány megállója van az új útvonalnak, majd az állomásokat. Ami nagyon fontos, hogy annak a vonatnak, amelynek megváltoztatjuk az útvonalát, annak törlődnek a jegyei is, mert ezek már nem aktuálisak.

6. **Menetrend megváltoztatása [6]:** Ki kell választani, hogy melyik vonat menetrendjét szeretnénk megváltoztatni, majd meg kell adni sorra az új indulás, illetve érkezés időpontját.

7. **Árak megváltoztatása [7]:** Miután kiválasztunk egy vonatot, kiválasztjuk, hogy melyik osztálynak szeretnénk megváltoztatni az árát és megadjuk az új árat.

8. **Vonatok rendezése szám szerint [8]:** Ezt az opciót választva a program rendezi a már felvett vonatokat szám szerint, és újra kilistázza őket.

9. **Vonatok rendezése dátum szerint [9]:** Ezt az opciót választva a program rendezi a már felvett vonatokat indulási időpontjuk szerint, és újra kilistázza őket.

10. **Vonatok szűrése típus szerint [10]:** Meg kell adnunk, hogy milyen típusú (intercity, helyközi) vonatokat szeretnénk kilistázni, majd csak a választott típusúak jelennek meg.

11. **Vonatok szűrése állomás szerint [11]:** Meg kell adnunk egy állomás nevét, majd azok a vonatok listázódnak ki, amelyek ezen az állomáson keresztülhaladnak.

12. **Összes vonat listázása [12]**

13. **Jegyek rendezése áruk szerint [13]:** Rendezi a jegyeket áruk szerinti növekvő sorrendben, majd kilistázza őket.

14. **Összes jegy listázása [14]**

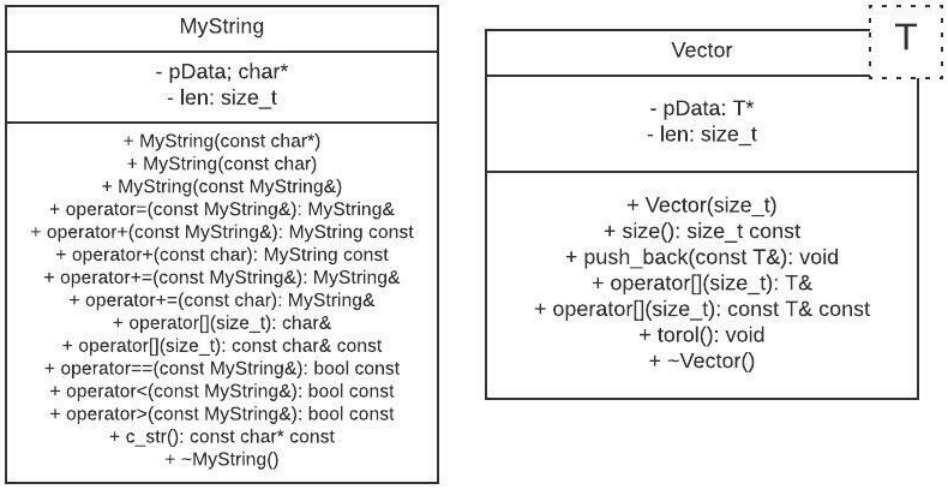
15. **Mentés [15]:** Ha elvégeztük az általunk kívánt módosításokat az adatbázison akkor ezt az opciót választva el lehet menteni a változtatásokat. Abban az esetben, ha előzőekben új adatbázist hoztunk létre, akkor mentés előtt a program fog kérni egy nevet, ami segítségével azonosítani tudjuk majd az adatbázist. Ennek a névnek nem szabad megegyeznie az eddig elmentett adatbázisok neveivel. Ha ez mégis megtörténik ezt a program jelezni fogja egy üzenet formájában. Mentés után újra az első menü fog megjelenni.

Terv

• Objektum terv

A karakterláncok tárolása érdekében létrehozok egy *MyString* osztályt, amely segítségével dinamikusan tárolom a karakterláncokat. Ennek köszönhetően tetszőleges hosszú karakterláncokkal tudok dolgozni a programomban. Ezen az osztályon a következő műveletek értelmezettek: értékadás, indexelés, egyenlőség vizsgálata, két string összefűzése, abc sorrend vizsgálata stb. (az osztálydiagramon látható).

Ahhoz, hogy tömböket tároljak létrehozok egy template *Vector* osztályt. Ebben dinamikusan lehet tárolni különböző típusú adatokat. Ezen az osztályon nem értelmezett az értékadás, emiatt privát. A *Vector* osztály elemei indexelhetőek, lekérdezhető, hogy éppen hány elemet tárolunk a vektorban, illetve új elemet lehet fűzni a vektor végére, vagy akár törölni is lehet az összes elemet belőle.



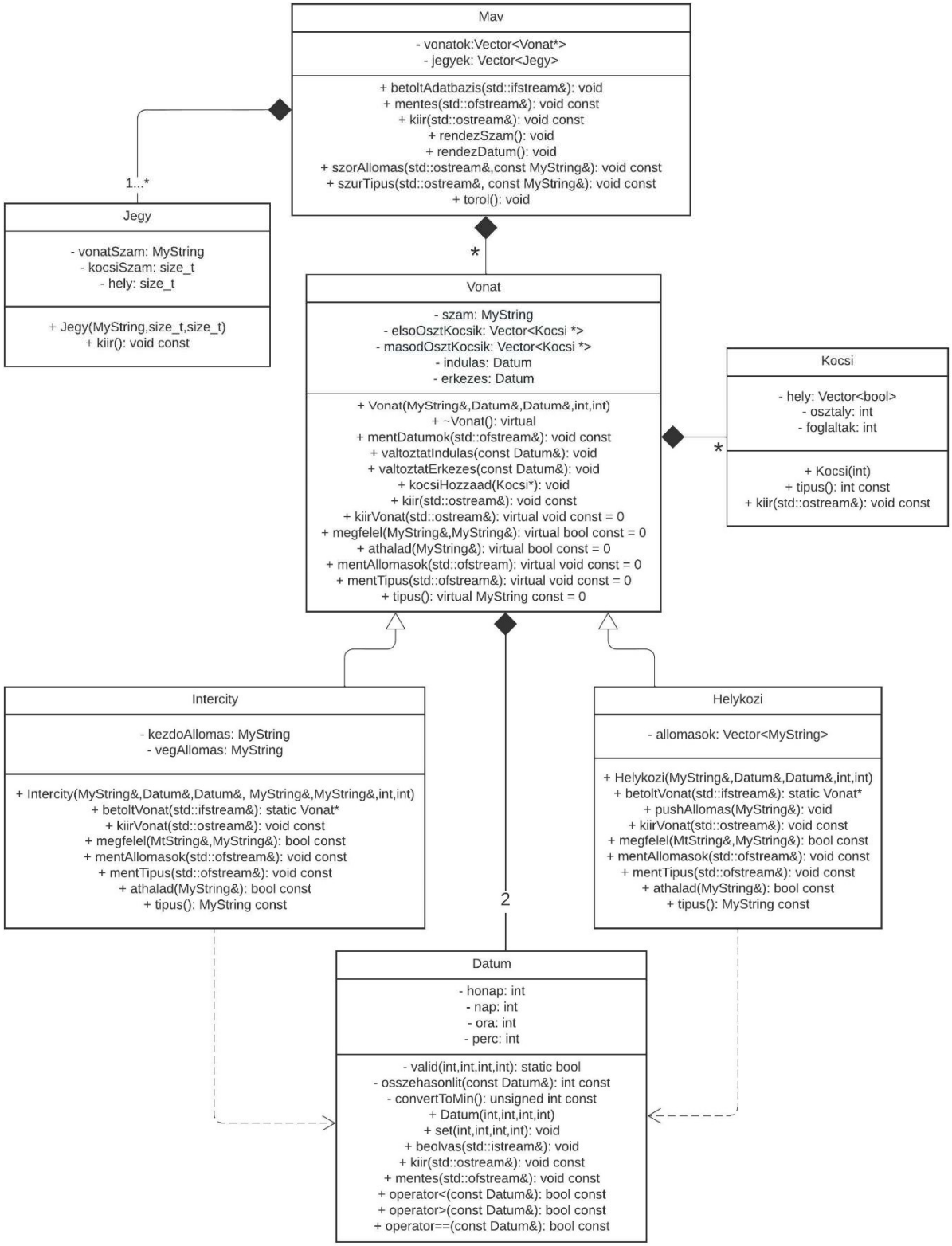
Létrehozok egy *Datum* osztályt is, amely segítségével egy időpontnak tudom tárolni a hónap, nap, óra és perc értékét. Az osztály egy adott dátumról le tudja ellenőrizni, hogy az helyes-e, össze lehet hasonlítani két dátumot, be lehet olvasni egy *std::istream*-ről egy dátumot, ki lehet íratni és el is lehet menteni egy fájlba.

A *Vonat* osztály a program egyik legfontosabb része. Ez egy absztrakt osztály, amelyből származnak az *Intercity* és *Helyközi* osztályok. A *Vonat* osztály segítségével tárolom egy adott vonat azonosító számát, a vonat kocsikra mutató *Kocsi** pointereket, a vonat indulási és érkezési időpontját. Menteni lehet egy fájlba az adott vonatot, vagy kiírni is lehet egy adott vonatot konzolba. Kocsik hozzáadására alkalmas metódusa is van az osztálynak, amellyel új kocsit lehet hozzáadni a vonathoz. Az egyik legfontosabb metódusa az osztálynak a *megfelel* tagfüggvénye, amellyel ellenőrzöm, hogy megfelel-e a vonat útvonala ahhoz az útvonalhoz, amit a felhasználó ad meg.

Az *Intercity* és *Helyközi* vonat között az a különbség, hogy míg az egyiknek két megállója van, ami két *MyString* típusú objektum, addig a másiknak több megállója van, így egy *MyString*-eket tároló *Vector* adattagja van. Az adatstruktúrák eltéréséből adódik, hogy más algoritmusokkal lehet kiírni, elmenteni, vagy meghatározni valamit a két különböző típussal kapcsolatban.

A *Mav* osztály tárolja a különböző vonatoknak a *Vonat** pointereit, egy heterogén kollekcióként. Egy *Mav* típusú objektumba be lehet tölteni egy adatbázist, ki lehet írni az összes vonatot, amit az adatbázis tartalmaz és különböző rendezéseket és szűréseket lehet a vonatokon végezni. Lényeges,

hogy a vonat pointerek egy *Vector* típusban vannak tárolva, tehát bármennyi vonatot tudunk egyszerre tárolni.



4.2. Algoritmusok

1. Helyközi vonat megfelel

Az algoritmusnak az a célja, hogy eldöntse, hogy egy adott helyközi vonatnak az útvonala megfelel vagy sem, a felhasználó által megadott útvonalhoz.

```
i = 0
amíg i < allomasok.size()
        ha allomasok[i] == kezdo, akkor
                amig i < allomasok.size()
                        ha allomasok[i] == veg, akkor
                                return true
                        ha vége
                        i = i + 1
                amíg vége
        ha vége
        i = i + 1
amíg vége
return false
```

1. Rendezés

A rendezést több alkalommal is kell használni a programban, és más típusú adatokra. Ezekben az adatokban az a közös, hogy egy *Vector* típusú objektumban vannak tárolva. Az algoritmusban szereplő *pred()* függvény egy predikátum, amelyben definiálva van, miszerint is kell rendezni a tömböt.

```
minden i=0, i<vec.size()
        minindex = i
        minden j=0, j<vec.size()
                ha pred(vec[j],vec[minindex]), akkor
                        minindex = j
                ha vége
        minden vége
        ha minindex != i
                swap(vec[i],vec[minindex])
        ha vége
minden vége
```

2. Bemeneti fájlok

Egy bemeneti fájl első sorában az adott adatbázisban szereplő vonatok száma található, ezután következik az n darab vonat: elsősorban egy 0-s vagy 1-es jelöli, hogy helyközi vagy intercity vonatról

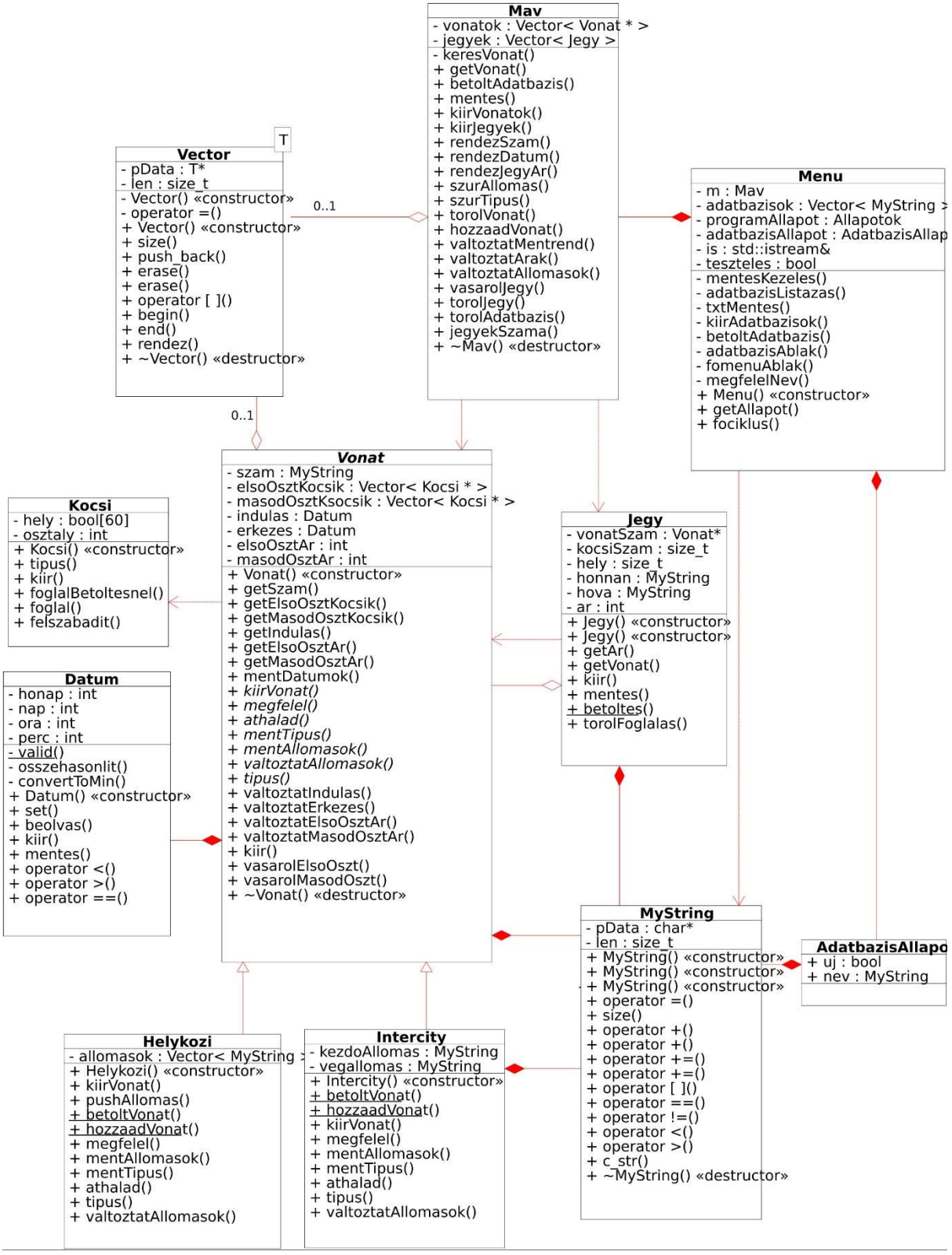
beszélünk, ezután következik a vonat azonosítószáma. A következő sorban van az első és a másodosztályú kocsik száma. Ezt követi új sorban az indulást és érkezést ábrázoló időpontok adatai. Intercity vonatok estén következik két különböző sorban kiindulási állomás, illetve célállomás. Helyközi vonatok estén előbb egy szám van, ami jelöli a megállók számát, majd egyenként a megállók. Egy lehetséges bementi fájl a következő lehet:

```
4

1 D3RJ9
3 4
2 12 22 10 2 13 10 15
Marosvasarhely
Budapest

0 C0HGI
2 8
2 12 22 10 2 13 10 15
5
Jaszbereny
Zala
Targu Mures
Budapest
Debrecen

0 B8IH
3 3
2 12 22 10 2 13 10 15
2
Nagyvarad
Szeben

1 AG7I
99 100
2 12 22 10 2 13 10 15
Jaszvasar
Bukarest
```

5.Megvalósítás

A feladat megvalósításához 11 darab osztályra volt szükségem, ezek a következőek: Menü, Máv, Vonat, Vector, Kocsi, Dátum, Jegy, Helykozi, Intercity, MyString, AdatbazisÁllapot. A tervhez képest több változtatást is végeztem az osztályokon, mint például az interfészükön, illetve új osztályokat is létrehoztam. A saját osztályok mellett használtam az std::exeption, illetve a teszteléshez használtam az std::stringstream osztályt.

5.1 Oszálydiagram



Osztályok dokumentációja

AdatbazisAllapot struktúrareferencia

Publikus attribútumok

- bool **uj**
- **MyString nev**

Részletes leírás

Az éppen betöltött adatbázis állapota. Ha az adatbázis nem új, akkor a **MyString** típusban az van tárolva, hogy mentéskor milyen nevű fájlba kell visszamenteni. Ha az adatbázis új, akkor a MyStringben tárolt adat érvénytelen, mert mentésnél a felhasználó adja meg, hogy mi legyen az adatbázis neve.

Adattagok dokumentációja

MyString AdatbazisAllapot::nev

bool AdatbazisAllapot::uj

Vector< T >::const_iterator osztályreferencia

Publikus tagfüggvények

- **const_iterator** ()
  default konstruktor

- **const_iterator** (const **Vector** &a, size_t ix=0)
- const T & **operator*** () const
  indirekció

- const T * **operator->** () const
  Szelekció pointerrel (nyíl). Címet kell, hogy adjon.

Részletes leírás

template<class T>

class Vector< T >::const_iterator

const_iterator osztály. Visszavezetjük az iterator osztályra.

Konstruktorok és destruktorok dokumentációja

template<class T > Vector< T >::const_iterator::const_iterator () [inline]

default konstruktor

template<class T > Vector< T >::const_iterator::const_iterator (const Vector & a, size_t ix = 0) [inline]

konstruktor, egy konkrét objektum ix-edik elemére mutat

Paraméterek

a	- a konkrét objektum
ix	- ide állítja az indexet

Tagfüggvények dokumentációja

template<class T > const T & Vector< T >::const_iterator::operator () const [inline]*

indirekció

Kis trükközés a kasztolással: Levesszük a konstans attribútumot, de nem fogjuk írni.

Ezen keresztül biztosan nem fogjuk írni, de sajnos kompatibilitás miatt egy **const_iterator** -> iterator konverzió nem kerül semmibe...

*template<class T > const T * Vector< T >::const_iterator::operator-> () const [inline]*

Szelekció pointerrel (nyíl). Címet kell, hogy adjon.

Datum osztályreferencia

Publikus tagfüggvények

- **Datum** (int **honap**=1, int **nap**=1, int **ora**=12, int **perc**=0)
  Konstruktor, amely egyben default konstruktor is amely első hónap, első nap, dél 12 órát állít be.

- void **set** (int **honap**, int **nap**, int **ora**, int **perc**)
- void **beolvas** (std::istream &is)
- void **kiir** (std::ostream &os) const
- void **mentes** (std::ostream &os) const
- bool **operator<** (const **Datum** &b) const
- bool **operator>** (const **Datum** &b) const
- bool **operator==** (const **Datum** &b) const

Privát tagfüggvények

- int **osszehasonlit** (const **Datum** &b) const
- unsigned int **convertToMin** () const

Statikus privát tagfüggvények

- static bool **valid** (int **honap**, int **nap**, int **ora**, int **perc**)

Privát attribútumok

- int **honap**
- int **nap**
- int **ora**
- int **perc**
  egész típusú változók a dátum adatainak tárolásához

Részletes leírás

Datum osztály. Egy dátum (hónap, nap) és idő (perc, másodperc) tárolására képes. Ha rossz dátumot adunk meg const char* kivételt dob.

Konstruktorok és destruktorok dokumentációja

Datum::Datum (int honap = 1, int nap = 1, int ora = 12, int perc = 0)

Konstruktor, amely egyben default konstruktor is, amely első hónap, első nap, dél 12 órát állít be.

Tagfüggvények dokumentációja

void Datum::beolvas (std::istream & is)

Dátum beolvasása.

Paraméterek

is	- input stream ahonnan olvas

unsigned int Datum::convertToMin () const [private]

Átváltás percre.

Visszatérési érték

- adott dátum percben számított értéke

void Datum::kiir (std::ostream & os) const

Dátum kiírása.

Paraméterek

os	- output stream ahova írunk

void Datum::mentes (std::ostream & os) const

Dátum mentése.

Paraméterek

os	- output file stream ahova mentünk

bool Datum::operator< (const Datum & b) const

Összehasonító operátor.

Paraméterek

b	- jobb oldali operandus

Visszatérési érték

- igaz, ha a bal oldali dátum hamarabb van

bool Datum::operator== (const Datum & b) const

Egyenlőség ellenőrzés operátor.

Paraméterek

b	- jobb oldali operandus

Visszatérési érték

- igaz, ha a két dátum egyenlõ

bool Datum::operator> (const Datum & b) const

Összehasonító operátor.

Paraméterek

b	- jobb oldali operandus

Visszatérési érték

- igaz, ha a jobb oldali dátum hamarabb van

int Datum::osszehasonlit (const Datum & b) const [private]

Dátumot összehasonlító függvény.

Paraméterek

b	- összehasonlítani kívánt dátum referenciája

Visszatérési érték

- 0 ha egyenlőek, -1 ha sorrendben vannak, 1 ha az első nagyobb

void Datum::set (int honap, int nap, int ora, int perc)

Setter függvény, amely ellenőrzi az adatok helyességét is.

Paraméterek

honap	- új hónap értéke
nap	- új nap értéke
ora	- új óra értéke
perc	- új perc értéke

bool Datum::valid (int honap, int nap, int ora, int perc) [static], [private]

Dátum helyesség ellenőrzésé

Paraméterek

honap	- dátum hónapja
nap	- dátum napja
ora	- időpont órája
perc	- időpont perce

Adattagok dokumentációja

int Datum::honap [private]

int Datum::nap [private]

int Datum::ora [private]

int Datum::perc [private]

egész típusú változók a dátum adatainak tárolásához

Helykozi osztályreferencia

Publikus tagfüggvények

- **Helykozi** (const **MyString** &**szam**, const **Datum** &**indulas**, const **Datum** &**erkezes**, int elsoAr, int masodAr, int elso=0, int masodik=0)
- void **kiirVonat** (std::ostream &os) const override
- void **pushAllomas** (const **MyString** &all)
- bool **megfelel** (const **MyString** &kezdo, const **MyString** &veg) const override
- void **mentAllomasok** (std::ostream &os) const override
- void **mentTipus** (std::ostream &os) const override

- bool **athalad** (const **MyString** &allomas) const override
- **MyString tipus** () const override
- void **valtoztatAllomasok** (std::istream &is) override

Statikus publikus tagfüggvények

- static **Vonat * betoltVonat** (std::ifstream &is)
- static **Vonat * hozzaadVonat** (std::istream &is)

Privát attribútumok

- **Vector< MyString > allomasok**
  ***Vector** típus, amelyben MyString-ként tároljuk a vonat megállóit (min 3 kell legyen)*

Részletes leírás

Helykozi osztály. **Vonat** osztályból van származtatva. Helyközi vonatokkal kapcsolatos műveleteket képes elvégezni.

Konstruktorok és destruktorok dokumentációja

Helykozi::Helykozi (const MyString & szam, const Datum & indulas, const Datum & erkezes, int elsoAr, int masodAr, int elso = 0, int masodik = 0)

Konstruktor,amelyben meg kell adni minden adatot a helyközi vonattal kapcsolatban.

Paraméterek

szam	- vonat azonosító száma
indulas	- vonat indulásának időpontja
erkezes	- vonat érkezésének időpontja
elsoAr	- első osztályon egy jegy ára
masodAr	- másod osztályon egy jegy ára
elso	- első osztályú kocsik száma
masodik	- másod osztályú kocsik ára

Tagfüggvények dokumentációja

bool Helykozi::athalad (const MyString & allomas) const `[override], [virtual]`

Eldönti, hogy a vonat áthalad-e vagy sem egy adott megállón.

Paraméterek

allomas	- megálló, amiről el kell dönteni, hogy az útvonal része-e

Visszatérési érték

igaz, ha a vonat áthalad a megadott megállón

*Vonat * Helykozi::betoltVonat (std::ifstream & is) `[static]`*

Statikus függvény egy helyközi vonat betöltéséhez.

Paraméterek

is	- input filestream, ahonnan be szeretnénk tölteni az adatokat

Visszatérési érték

dinamikusan foglalt vonat mutató

Figyelmeztetés

- a függvény dinamikusan foglal le egy helyközi vonatot, a hívó felelőssége ezt felszabadítani

*Vonat * Helykozi::hozzaadVonat (std::istream & is) [static]*

Statikus függvény egy helyközi vonat hozzáadásához az adatbázishoz.

Paraméterek

is	- input stream, ahonnan be szeretnénk vinni a vonat adatait

Visszatérési érték

dinamikusan foglalt vonat mutató

Figyelmeztetés

- a függvény dinamikusan foglal le egy helyközi vonatot, a hívó felelőssége ezt felszabadítani

void Helykozi::kiirVonat (std::ostream & os) const [override], [virtual]

Egy helyközi vonat kiírására szolgáló függvény.

Paraméterek

os	- output stream, ahova ki szeretnénk írni az adatokat

bool Helykozi::megfelel (const MyString & kezdo, const MyString & veg) const [override], [virtual]

Eldönti, egy adott kezdő- és végállomásról, hogy a vonat útvonalának része-e vagy nem.

Paraméterek

kezdo	- kezdő állomás
veg	- végállomás

Visszatérési érték

igaz, ha a vonat megfelel az útvonalnak

void Helykozi::mentAllomasok (std::ostream & os) const [override], [virtual]

Elmenti az állomásait egy adott output streambe.

Paraméterek

os	- output stream ahova szeretnénk menteni az állomásokat

void Helykozi::mentTipus (std::ostream & os) const [override], [virtual]

Elmenti a helyközi vonat típusát.

Paraméterek

os	- output stream ahova a típust szeretnénk menteni

void Helykozi::pushAllomas (const MyString & all)

Egy újabb állomást ad hozzá az eddigi útvonalhoz.

Paraméterek

all	- az új állomás neve

MyString Helykozi::tipus () const [override], [virtual]

Egy MyStringet ad vissza, amiben a vonat típusa van, azaz Helyközi.

Visszatérési érték

MyString típus, amiben "Helykozi" van

void Helykozi::valtoztatAllomasok (std::istream & is) [override], [virtual]

A vonat állomásainak a megváltoztatására szolgáló függvény.

Paraméterek

is	- input stream, ahonnan jönnek az szükséges adatok

Visszatérési érték

kivételt dob, ha kevesebb mint 3 megállót akarunk megadni

Adattagok dokumentációja

Vector<MyString> Helykozi::allomasok [private]

Vector típus, amelyben MyString-ként tároljuk a vonat megállóit (min 3 kell legyen)

Intercity osztályreferencia

Publikus tagfüggvények

- **Intercity** (const **MyString** &**szam**, const **Datum** &**indulas**, const **Datum** &**erkezes**, const **MyString** &kezdo, const **MyString** &veg, int elsoAr, int masodAr, int elso=1, int masodik=0)
- void **kiirVonat** (std::ostream &os) const override
- bool **megfelel** (const **MyString** &kezdo, const **MyString** &veg) const override
- void **mentAllomasok** (std::ostream &os) const override
- void **mentTipus** (std::ostream &os) const override
- bool **athalad** (const **MyString** &allomas) const override
- **MyString tipus** () const override
- void **valtoztatAllomasok** (std::istream &is) override

Statikus publikus tagfüggvények

- static **Vonat** * **betoltVonat** (std::ifstream &is)
- static **Vonat** * **hozzaadVonat** (std::istream &is)

Privát attribútumok

- **MyString kezdoAllomas**
  MyString *típus, amelyben az kezdő állomás van tarolva.*

- **MyString vegallomas**
  MyString *típus, amelyben az végállomás van tarolva.*

Részletes leírás

Intercity osztály. **Vonat** osztályból van származtatva. **Intercity** vonatokkal kapcsolatos műveleteket képes elvégezni.

Konstruktorok és destruktorok dokumentációja

Intercity::Intercity (const MyString & szam, const Datum & indulas, const Datum & erkezes, const MyString & kezdo, const MyString & veg, int elsoAr, int masodAr, int elso = 1, int masodik = 0)

Konstruktor, amelyben meg kell adni minden adatot az intercity vonattal kapcsolatban.

Paraméterek

szam	- vonat azonosító száma
indulas	- vonat indulásának időpontja
erkezes	- vonat érkezésének időpontja
kezdo	- a vonat kezdő állomása
veg	- a vonat végállomása
elsoAr	- első osztályon egy jegy ára
masodAr	- másod osztályon egy jegy ára
elso	- első osztályú kocsik száma
masodik	- másod osztályú kocsik ára

Tagfüggvények dokumentációja

bool Intercity::athalad (const MyString & allomas) const [override], [virtual]

Eldönti, hogy a vonat áthalad-e vagy sem egy adott megállón.

Paraméterek

allomas	- megálló, amiről el kell dönteni, hogy az útvonal része-e

Visszatérési érték

igaz, ha a vonat áthalad a megadott megállón

*Vonat * Intercity::betoltVonat (std::ifstream & is) [static]*

Statikus függvény egy intercity vonat betöltéséhez.

Paraméterek

is	- input filestream, ahonnan be szeretnénk tölteni az adatokat

Visszatérési érték

dinamikusan foglalt vonat mutató

Figyelmeztetés

- a függvény dinamikusan foglal le egy intercity vonatot, a hívó felelőssége ezt felszabadítani

*Vonat * Intercity::hozzaadVonat (std::istream & is) [static]*

Statikus függvény egy intercity vonat hozzáadásához az adatbázishoz.

Paraméterek

is	- input stream, ahonnan be szeretnénk vinni a vonat adatait

Visszatérési érték

dinamikusan foglalt vonat mutató

Figyelmeztetés

- a függvény dinamikusan foglal le egy intercity vonatot, a hívó felelőssége ezt felszabadítani

void Intercity::kiirVonat (std::ostream & os) const [override], [virtual]

Egy intercity vonat kiírására szolgáló függvény.

Paraméterek

os	- output stream, ahova ki szeretnénk írni az adatokat

bool Intercity::megfelel (const MyString & kezdo, const MyString & veg) const [override], [virtual]

Eldönti, egy adott kezdő- és végállomásról, hogy ugyanazok-e, mint a vonat állomásai.

Paraméterek

kezdo	- kezdő állomás
veg	- végállomás

Visszatérési érték

igaz ha a vonat megfelel az útvonalnak

void Intercity::mentAllomasok (std::ostream & os) const [override], [virtual]

Elmenti az állomásait egy adott output streambe.

Paraméterek

os	- output stream ahova szeretnénk menteni az állomásokat

void Intercity::mentTipus (std::ostream & os) const [override], [virtual]

Elmenti az intercity vonat típusát.

Paraméterek

os	- output stream ahova a típust szeretnénk menteni

MyString Intercity::tipus () const [override], [virtual]

Egy MyStringet ad vissza, amiben a vonat típusa van, azaz **Intercity**.

Visszatérési érték

MyString típus, amiben "Intercity" van

void Intercity::valtoztatAllomasok (std::istream & is) [override], [virtual]

A vonat állomásainak a megváltoztatására szolgáló függvény.

Paraméterek

is	- input stream, ahonnan jönnek az szükséges adatok

Adattagok dokumentációja

MyString Intercity::kezdoAllomas [private]

MyString típus, amelyben az kezdõállomás van tarolva.

MyString Intercity::vegallomas [private]

MyString típus, amelyben az végállomás van tarolva.

Vector< T >::iterator osztályreferencia

Publikus tagfüggvények

- **iterator** ()
  Default konstruktor.

- **iterator** (**Vector** &a, size_t ix=0)
- size_t **index** () const
- **iterator** & **operator++** ()
- **iterator** & **operator--** ()
- **iterator operator++** (int)
- bool **operator!=** (const **iterator** &i) const
- bool **operator==** (const **iterator** &i) const
- T & **operator*** () const
  indirekció

- T * **operator->** () const
  Szelekció pointerrel (nyíl). Címet kell, hogy adjon.

Privát attribútumok

- **Vector * p**
  tároljuk, hogy melyik az objektumhoz tartozik az iterátor

- size_t **idx**
  tároljuk az aktuális index értéket

Részletes leírás

template<class T>

class Vector< T >::iterator

Iterator osztály.

Konstruktorok és destruktorok dokumentációja

template<class T > Vector< T >::iterator::iterator () [inline]

Default konstruktor.

template<class T > Vector< T >::iterator::iterator (Vector & a, size_t ix = 0) [inline]

Konstruktor, egy konkrét objektum ix-edik elemére mutat.

Paraméterek

a	- a konkrét objektum
ix	- ide állítja az indexet

Tagfüggvények dokumentációja

template<class T > size_t Vector< T >::iterator::index () const [inline]

Tagfüggvény mely visszaadja, hogy az iterátor, hányadik elemre mutat.

Visszatérési érték

index

template<class T > bool Vector< T >::iterator::operator!= (const iterator & i) const [inline]

Egyenlőtlenség vizsgálat.

Paraméterek

i	- jobboldali operandus

Visszatérési érték

igaz, ha nem egyenlőek az iterátorok által mutatott adatok

template<class T > T & Vector< T >::iterator::operator () const [inline]*

indirekció

template<class T > iterator & Vector< T >::iterator::operator++ () [inline]

Pre-inkremens operátor. Csak hatékonyság miatt ref. visszatérésű, értelmetlen lenne balértékként használni.

Visszatérési érték

léptett iterator

template<class T > iterator Vector< T >::iterator::operator++ (int) [inline]

Post-inkremens operátor.

Visszatérési érték

léptet iterator

template<class T > iterator & Vector< T >::iterator::operator-- () [inline]

Pre-dekremens. Csak hatékonyság miatt ref. visszatérésű, értelmetlen lenne balértékként használni

Visszatérési érték

léptett iterator

*template<class T > T * Vector< T >::iterator::operator-> () const [inline]*

Szelekció pointerrel (nyíl). Címet kell, hogy adjon.

template<class T > bool Vector< T >::iterator::operator== (const iterator & i) const [inline]

Egyenlőség vizsgálat.

Paraméterek

i	- jobboldali operandus

Visszatérési érték

igaz, ha egyenlőek az iterátorok által mutatott adatok

Adattagok dokumentációja

template<class T > size_t Vector< T >::iterator::idx [private]

tároljuk az aktuális index értéket

template<class T > Vector Vector< T >::iterator::p [private]*

tároljuk, hogy melyik az objektumhoz tartozik az iterátor

Jegy osztályreferencia

Publikus tagfüggvények

- **Jegy** ()
  Default konstruktor.

- **Jegy** (**Vonat** *vonatszam, size_t **kocsiSzam**, size_t **hely**, const **MyString** &**honnan**, const **MyString** &**hova**, int **ar**)
  Konstrukor amelyben meg kell adni az összes jeggyel kapcsolatos adatot.

- int **getAr** () const
- **Vonat** * **getVonat** () const
- void **kiir** (std::ostream &os) const
- void **mentes** (std::ostream &os) const
- void **torolFoglalas** () const
  A függvény az adott jegy adattagjaiban megadott információk szerint törli a foglalást, azaz újra szabad lesz a vonaton ez a hely.

Statikus publikus tagfüggvények

- static **Jegy** **betoltes** (std::ifstream &is, **Vector**< **Vonat** * > &vonatok)

Privát attribútumok

- **Vonat** * **vonatSzam**
  vonatnak a száma amire szól a jegy

- size_t **kocsiSzam**
  vonaton belüli kocsi száma

- size_t **hely**
  a kocsin belül a hely száma

- **MyString** **honnan**
  honnan indul a vonat

- **MyString** **hova**
  hova érkezik

- int **ar**
  jegy ára

Konstruktorok és destruktorok dokumentációja

Jegy::Jegy () [inline]

Default konstruktor.

*Jegy::Jegy (Vonat * vonatszam, size_t kocsiSzam, size_t hely, const MyString & honnan, const MyString & hova, int ar)*

Konstrukor amelyben meg kell adni az összes jeggyel kapcsolatos adatot.

Tagfüggvények dokumentációja

*Jegy Jegy::betoltes (std::ifstream & is, Vector< Vonat * > & vonatok) [static]*

Létrehoz egy **Jegy** típusú objektumot és feltölti adatokkal.

Paraméterek

is	- input file stream ahonnan olvassa az adatokat a jegybe

Visszatérési érték

- a létrehozott **Jegy** típusú objektum

int Jegy::getAr () const

Jegy árának lekérdezése.

Visszatérési érték

- az adott jegynek az ára

*Vonat * Jegy::getVonat () const*

Megadja, hogy melyik vonatra van a jegy.

Visszatérési érték

- adott jegyhez tartozó vonat

void Jegy::kiir (std::ostream & os) const

Kiírja a jegy adatait.

Paraméterek

os	- output stream ahova kiírja a jegy adatait

void Jegy::mentes (std::ostream & os) const

Elmenti a jegy adatait.

Paraméterek

os	- output file stream ahova elmenti a jegy adatait

void Jegy::torolFoglalas () const

A függvény az adott jegy adattagjaiban megadott információk szerint törli a foglalást, azaz újra szabad lesz a vonaton ez a hely.

Adattagok dokumentációja

int Jegy::ar [private]

jegy ára

size_t Jegy::hely [private]

a kocsin belül a hely száma

MyString Jegy::honnan [private]

honnan indul a vonat

MyString Jegy::hova [private]

hova érkezik

size_t Jegy::kocsiSzam [private]

vonaton belüli kocsi száma

Vonat Jegy::vonatSzam [private]*

vonatnak a száma amire szól a jegy

Kocsi osztályreferencia

Publikus tagfüggvények

- **Kocsi** (int **tipus**=1)
  Konstruktor amelyben meg lehet adni a kocsi osztályát, de default konstruktor is egyben.

- int **tipus** () const
  Getter függvény a kocsi típusának a lekérdezéséhez.

- void **kiir** (std::ostream &os) const
  ***Kocsi** kiírása.*

- void **foglalBetoltesnel** (size_t sorszam)
- int **foglal** ()
- void **felszabadit** (size_t sorszam)

Privát attribútumok

- bool **hely** [60]
  az adott kocsiban lévő helyek állapota

- int **osztaly**
  a kocsi osztályának a típusa: első vagy második osztály

Részletes leírás

Kocsi osztály: egy vonatnak ilyen típusúak a kocsikból áll.

Konstruktorok és destruktorok dokumentációja

Kocsi::Kocsi (int tipus = 1)

Konstruktor amelyben meg lehet adni a kocsi osztályát, de default konstruktor is egyben.

Tagfüggvények dokumentációja

void Kocsi::felszabadit (size_t sorszam)

Újból szabaddá tesz egy adott helyet a kocsin belül.

Paraméterek

sorszam	- a felszabadítani kívánt hely sorszáma

int Kocsi::foglal ()

A függvény megkeresi az első szabad helyet a kocsin belül és lefoglalja, majd visszaadja a hely sorszámát. Ha a kocsi már tele van, azaz nincs üres hely benne, akkor -1 értéket ad vissza

Visszatérési érték

a hely sorszáma vagy -1 ha nincs üres hely

void Kocsi::foglalBetoltesnel (size_t sorszam)

A függvény egy adatbázis betöltésénél használandó, segítségével le tudunk foglalni a kocsiban egy adott helyet.

Paraméterek

sorszam	- a kocsin belüli hely sorszáma, amit le akarunk foglalni

void Kocsi::kiir (std::ostream & os) const

Kocsi kiírása.

int Kocsi::tipus () const

Getter függvény a kocsi típusának a lekérdezéséhez.

Adattagok dokumentációja

bool Kocsi::hely[60] [private]

az adott kocsiban lévő helyek állapota

int Kocsi::osztaly [private]

a kocsi osztályának a típusa: első vagy második osztály

Mav osztályreferencia

Publikus tagfüggvények

- **Vonat * getVonat** (size_t i) const
- void **betoltAdatbazis** (std::ifstream &is)
- void **mentes** (std::ostream &os) const
- void **kiirVonatok** (std::ostream &os=std::cout) const
- void **kiirJegyek** (std::ostream &os=std::cout) const
- void **rendezSzam** ()
  Rendezi a vonatokat a tárolójukban szám szerint növekvő sorrendben.

- void **rendezDatum** ()
  Rendezi a vonatokat a tárolójukban dátum szerint növekvő sorrendben.

- void **rendezJegyAr** ()
  Rendezi a jegyeket az áruk szerint növekvő sorrendben.

- int **szurAllomas** (std::istream &is, std::ostream &os=std::cout) const
- int **szurTipus** (std::istream &is, std::ostream &os=std::cout) const
- void **torolVonat** (std::istream &is)
- void **hozzaadVonat** (std::istream &is)
- void **valtoztatMentrend** (std::istream &is)
- void **valtoztatArak** (std::istream &is)
- void **valtoztatAllomasok** (std::istream &is)
- void **vasarolJegy** (std::istream &is)
- void **torolJegy** (std::istream &is)

- void **torolAdatbazis** ()
  Törli a dinamikus foglalt vonatokat, felszabadítja a két vektornak lefoglalt helyet is.

- size_t **jegyekSzama** () const
- **~Mav** ()
  Destruktor.

Privát tagfüggvények

- **Vonat * keresVonat** (const **MyString** &szam)

Privát attribútumok

- **Vector< Vonat * > vonatok**
  ***Vector** típusú objektum, az adott adatbázishoz tartozó vonatok Vonat* mutatói.*

- **Vector< Jegy > jegyek**
  ***Vector** típusú objektum, amelyben a jegyek van tárolva.*

Részletes leírás

Mav osztály. Ebben az osztályban vannak tárolva a vonatok, illetve a jegyek. Ez az osztály teremti meg a kapcsolatot a felhasználó és a módosításokat végző **Vonat** és **Jegy** osztályok között.

Konstruktorok és destruktorok dokumentációja

Mav::~Mav ()

Destruktor.

Tagfüggvények dokumentációja

void Mav::betoltAdatbazis (std::ifstream & is)

Betölti egy adatbázis elemeit egy adott input file streamből.

Paraméterek

is	- input file stream amiből az adatokat kell betölteni

*Vonat * Mav::getVonat (size_t i) const*

Visszaadja egy adott sorszámú vonat mutatóját.

Paraméterek

i	- hányadik vonat mutatóját szeretnénk

Visszatérési érték

az adott sorszámú vonatra mutató pointer

void Mav::hozzaadVonat (std::istream & is)

Egy új vonat hozzáadását bonyolítja le. Ha nem megfelelő adatokat adunk meg, akkor kivételt dob.

Paraméterek

is	- input stream, ahonnan érkeznek az adatok

size_t Mav::jegyekSzama () const

Getter függvény a jegyek számának lekérdezésére.

Visszatérési érték

az adatbázisban lévő jegyek száma

*Vonat * Mav::keresVonat (const MyString & szam) [private]*

void Mav::kiirJegyek (std::ostream & os = std::cout) const

Kiírja az összes jegy adatait egy adott output streambe.

Paraméterek

os	- output stream ahova a jegyeket szeretnénk kiírni

void Mav::kiirVonatok (std::ostream & os = std::cout) const

Kiírja az összes vonat adatatit egy adott output streambe.

Paraméterek

os	- ouput stream ahova a vonatokat szeretnénk kiírni

void Mav::mentes (std::ostream & os) const

Elmenti a vonatok, illetve jegyek adatatit egy adott output streambe.

Paraméterek

os	- output stream ahova az adatokat szeretnénk menteni

void Mav::rendezDatum ()

Rendezi a vonatokat a tárolójukban dátum szerint növekvő sorrendben.

void Mav::rendezJegyAr ()

Rendezi a jegyeket az áruk szerint növekvő sorrendben.

void Mav::rendezSzam ()

Rendezi a vonatokat a tárolójukban szám szerint növekvő sorrendben.

int Mav::szurAllomas (std::istream & is, std::ostream & os = std::cout) const

Egy szűrést végez a vonatokra egy megadott állomás szerint, és kiírja a feltételnek megfelelő vonatokat. Ha nincs megfelelő bejegyzés, akkor kivételt dob.

Paraméterek

is	- input stream ahonnan, a függvény az állomás nevét kapja, ami szerint a szűrést végzi
os	- output stream ahova, a függvény kiírja a feltételnek megfelelő bejegyzéseket

int Mav::szurTipus (std::istream & is, std::ostream & os = std::cout) const

Egy szűrést végez a vonatokra egy megadott típus szerint, és kiírja a feltételnek megfelelő vonatokat. Ha nincs megfelelő bejegyzés, akkor kivételt dob.

Paraméterek

is	- input stream, ahonnan a függvény az típus nevét kapja, ami szerint a szűrést végzi.

os	- output stream, ahova a függvény kiírja a feltételnek megfelelő bejegyzéseket

void Mav::torolAdatbazis ()

Törli a dinamikus foglalt vonatokat, felszabadítja a két vektornak lefoglalt helyet is.

void Mav::torolJegy (std::istream & is)

Jegy törlését végzi. Elsősorban kilistázza az adatbázisban szereplő jegyeket, majd egy sorszámot megadva törli az adott jegyet. Ha érvénytelen sorszámot kap, akkor kivételt dob.

Paraméterek

is	- input stream, ahonnan az adatok érkeznek

void Mav::torolVonat (std::istream & is)

Kitöröl egy felhasználó által megadott vonatot és a vonatra tartozó jegyeket. Ha nincs olyan vonat az adatbázisban, mint amit a felhasználó megad, akkor kivételt dob.

Paraméterek

is	- input stream, ahonnan megkapja, a törölni kívánt vonat azonosítószámát

void Mav::valtoztatAllomasok (std::istream & is)

Egy adatbázisban lévő vonat állimásainak módosítását bonyolítja le. Ha rossz azonosítószámot kap, akkor kivételt dob.

Paraméterek

is	- input stream, ahonnan érkeznek az adatok

void Mav::valtoztatArak (std::istream & is)

Egy adatbázisban lévő vonat árainak megváltoztatását kezeli. Ha rossz azonosítószámot kap, vagy rossz osztály lesz kiválasztva, akkor kivételt dob.

Paraméterek

is	- input stream ahonnan érkeznek az adatok

void Mav::valtoztatMentrend (std::istream & is)

Egy adatbázisban lévő vonat indulási és érkezési időpontjainak megváltoztatását kezeli. Ha olyan azonosítószámot kap, ami nem létezik, akkor kivételt dob.

Paraméterek

is	- input stream, ahonnan érkeznek az adatok

void Mav::vasarolJegy (std::istream & is)

Egy új jegy hozzáadását bonyolítja le. Ha nincs az általunk megadott útvonalnak megfelelő vonat, vagy rossz vonat számot adunk meg kiválasztásnál, akkor kivételt dob.

Paraméterek

is	- input stream, ahonnan az adatok érkeznek

Adattagok dokumentációja

Vector<Jegy> Mav::jegyek `[private]`

Vector típusú objektum, amelyben a jegyek van tárolva.

*Vector<Vonat *> Mav::vonatok [private]*

Vector típusú objektum, az adott adatbázishoz tartozó vonatok Vonat* mutatói.

Menu osztályreferencia

Publikus tagfüggvények

- **Menu** (std::istream &**is**=std::cin, bool **teszteles**=false)
- **Allapotok getAllapot** () const
- void **fociklus** ()
  Kezeli, hogy éppen melyik ablakot kell megjeleníteni, a programAllapot változótól függően, leállítja a programot szükség esetén, illetve kezeli a dobott kivételeket.

Privát tagfüggvények

- void **mentesKezeles** ()
- void **adatbazisListazas** ()
  Kilistázza a meglévõ adatbázisokat, majd kéri, hogy válasszuk ki, hogy melyiket szeretnénk betölteni és betölti azt az adatbázist.

- void **txtMentes** ()
- void **kiirAdatbazisok** () const
  Kiírja a létező adatbázisok nevét.

- void **betoltAdatbazis** (size_t szam)
- void **adatbazisAblak** ()
- void **fomenuAblak** ()
- bool **megfelelNev** (const **MyString** &nev) const

Privát attribútumok

- **Mav m**
  ***Mav** típusú objektum, amelyben tárolva vannak az adatbázis vonatjai és jegyei.*

- **Vector**< **MyString** > **adatbazisok**
  ***Vector** típusú objektum, melyben az adatbázisok fájlnevei vannak tárolva.*

- **Allapotok programAllapot**
  program nézete

- **AdatbazisAllapot adatbazisAllapot**
  az éppen betöltve lévő adatbázisnak a típusa

- std::istream & **is**
  input file stream ahonnan a programba bemenő adatok érkeznek

- bool **teszteles**
  milyen célból futtatjuk a programot: tesztelés vagy nem

Részletes leírás

Menu osztály. Ez az osztály felelős a program nézeteinek a kezeléséért, az adatbázisok kezeléséért, illetve a **Mav** osztály különböző osztályainak meghívásáért.

Konstruktorok és destruktorok dokumentációja

Menu::Menu (std::istream & is = `std::cin`, bool teszteles = `false`)

Menu osztály konstruktora.

Paraméterek

is	- input stream ahonnan érkeznek az utasítások és az adatok
teszteles	- milyen célból futtatjuk a programot: tesztelés vagy nem

---

Tagfüggvények dokumentációja

void Menu::adatbazisAblak () `[private]`

Az adatbázis ablakot kezeli. Kiírja a menü alpontjait, illetve egy adott utasítás sorszám esetén meghívja az ahhoz tartozó függvényt. Ha rossz utasítás kódja, akkor kivételt dob.

void Menu::adatbazisListazas () `[private]`

Kilistázza a meglévő adatbázisokat, majd kéri hogy válasszuk ki, hogy melyiket szeretnénk betölteni és betölti azt az adatbázist.

void Menu::betoltAdatbazis (size_t szam) `[private]`

Betölti a megadott sorszámú adatbázist. Ha rossz sorszámot adunk meg a kiválasztásnál akkor egy kivételt dob.

Paraméterek

szam	- a betölteni kívánt adatbázis sorszáma.

void Menu::fociklus ()

Kezeli, hogy éppen melyik ablakot kell megjeleníteni, a programAllapot változótól függően, leállítja a programot szükség esetén, illetve kezeli a dobott kivételeket.

void Menu::fomenuAblak () `[private]`

Az főmenü ablakot kezeli. Kiírja a menü alpontjait, illetve egy adott utasítás sorszám esetén meghívja az ahhoz tartozó függvényt. Ha rossz utasítás kódja, akkor kivételt dob.

Allapotok Menu::getAllapot () const

Getter függvény, amely visszaadja a menü állapotát.

Visszatérési érték

menü állapota

void Menu::kiirAdatbazisok () const `[private]`

Kiírja a létező adatbázisok nevét.

bool Menu::megfelelNev (const MyString & nev) const `[private]`

Azt ellenőrzi, hogy a paraméterben megadott név szerepel-e a már meglévő adatbázisok nevei között.

Paraméterek

nev	- a keresett adatbázis név

Visszatérési érték

igaz, ha nem szerepel olyan név, amit megadtunk paraméterben

void Menu::mentesKezeles () [private]

Abban az esetben, ha el akarunk menteni egy adatbázist ez a függvény lesz meghívva meg. Ha az elmenteni kívánt adatbázis új, akkor meg kell adni az új adatbázis nevét. Ha a megadott név alatt már szerepel adatbázis akkor kivételt dob a függvény. Ellenkező esetben ugyan abba a fájlba mentőik az adatbázis, ahonnan betöltöttük.

void Menu::txtMentes () [private]

A program folderében lévő .txt fájlokat menti el egy **Vector** típusú objektumba. Ha tesztelés céljából futtatunk, akkor a teszt állományait tölti be.

Adattagok dokumentációja

AdatbazisAllapot Menu::adatbazisAllapot [private]

az éppen betöltve lévő adatbázisnak a típusa

Vector<MyString> Menu::adatbazisok [private]

Vector típusú objektum, melyben az adatbázisok fájlnevei vannak tárolva.

std::istream& Menu::is [private]

input file stream ahonnan a programba bemenő adatok érkeznek

Mav Menu::m [private]

Mav típusú objektum, amelyben tárolva vannak az adatbázis vonatjai és jegyei.

Allapotok Menu::programAllapot [private]

program nézete

bool Menu::teszteles [private]

milyen célból futtatjuk a programot: tesztelés vagy nem

MyString osztályreferencia

Publikus tagfüggvények

- **MyString** (const char *str="")
- **MyString** (const char c)
- **MyString** (const **MyString** &s)
- **MyString** & **operator=** (const **MyString** &s)
- size_t **size** () const
- **MyString operator+** (const **MyString** &s) const
- **MyString operator+** (const char c) const
- **MyString** & **operator+=** (const **MyString** &s)
- **MyString** & **operator+=** (const char c)
- char & **operator[]** (size_t index)
- const char & **operator[]** (size_t index) const

- bool **operator==** (const **MyString** &s) const
- bool **operator!=** (const **MyString** &s) const
- bool **operator<** (const **MyString** &s) const
- bool **operator>** (const **MyString** &s) const
- const char * **c_str** () const
- **~MyString** ()
  Destruktor.

Privát attribútumok

- char * **pData**
  pointer az adatra

- size_t **len**
  hossz lezáró nulla nélkül

Részletes leírás

MyString osztály. A pData-ban vannak a karakterek (a lezáró nullával együtt). len a hossz. A hosszba nem számít bele a lezáró nulla.

Konstruktorok és destruktorok dokumentációja

*MyString::MyString (const char * str = "")*

Konstruktor egy nullával lezárt char sorozatból. Ez a deafault is!

Paraméterek

str	- pointer egy C sztringre

MyString::MyString (const char c)

Konstruktor egy char karakterből.

Paraméterek

c	- karakter

MyString::MyString (const MyString & s)

Másoló konstruktor.

Paraméterek

s	- **MyString**, amiből létrehozzuk az új String-et

MyString::~MyString ()

Destruktor.

Tagfüggvények dokumentációja

*const char * MyString::c_str () const*

C-sztringet ad vissza.

Visszatérési érték

pointer egy '\0'-val lezárt (C) sztringre

bool MyString::operator!= (const MyString & s) const

Összehasonlító operátor.

Paraméterek

s-	jobb oldali **MyString**

Visszatérési érték

-igaz, ha a két sztring különbözik

MyString MyString::operator+ (const char c) const

Sztringhez karaktert összefűz.

Paraméterek

c	- jobboldali karakter

Visszatérési érték

új **MyString**, ami tartalmazza a sztringet és a karaktert egymás után

MyString MyString::operator+ (const MyString & s) const

Két Stringet összefûz.

Paraméterek

s	- jobboldali String

Visszatérési érték

új **MyString**, ami tartalmazza a két stringet egymás után

MyString & MyString::operator+= (const char c)

Hozzáfűz egy karaktert az adott objektum végéhez.

Paraméterek

c	- karakter amit hozzá szeretnénk fűzni

Visszatérési érték

- MyString referencia az összefűzött sztringekre

MyString & MyString::operator+= (const MyString & s)

Hozzáfűz egy MyStringet az adott objektum végéhez.

Paraméterek

s	- **MyString** amit hozzá szeretnénk fűzni

Visszatérési érték

- MyString referencia az összefűzőt sztringekre

bool MyString::operator< (const MyString & s) const

Összehasonlító operátor.

Paraméterek

s-	jobb oldali **MyString**

Visszatérési érték

-igaz ha a bal oldal előbbre van az abc sorrendben

MyString & MyString::operator= (const MyString & s)

Értékadó operátor.

Paraméterek

s	- jobboldali String

Visszatérési érték

baloldali (módosított) string (referenciája)

bool MyString::operator== (const MyString & s) const

Összehasonlító operátor.

Paraméterek

s-	jobb oldali **MyString**

Visszatérési érték

-igaz ha a két sztring egyenlő

bool MyString::operator> (const MyString & s) const

Összehasonlító operátor.

Paraméterek

s-	jobb oldali **MyString**

Visszatérési érték

-igaz ha a bal oldal hátrébb van az abc sorrendben

char & MyString::operator[] (size_t index)

A string egy megadott indexű elemének referenciájával tér vissza.

Paraméterek

index	- karakter indexe

Visszatérési érték

karakter (referencia) Indexelési hiba esetén const char* kivételt dob.

const char & MyString::operator[] (size_t index) const

A string egy megadott indexű elemének referenciájával tér vissza.

Paraméterek

index	- karakter indexe

Visszatérési érték

karakter (referencia) Indexelési hiba esetén const char* kivételt dob (assert helyett).

size_t MyString::size () const

Milyen hosszú az adott **MyString** '\0' nélkül.

Visszatérési érték

MyString hossza

Adattagok dokumentációja

size_t MyString::len `[private]`

hossz lezáró nulla nélkül

char MyString::pData `[private]`*

pointer az adatra

Vector< T > osztálysablon-referencia

Osztályok

- class **const_iterator**
- class **iterator**
  Iterator osztály.

Publikus tagfüggvények

- **Vector** (size_t n=0)
- size_t **size** () const
- void **push_back** (const T &data)
- void **erase** ()
  Felszabadítja a vektor összes elemét.

- void **erase** (**iterator** elem)
- T & **operator[]** (size_t index)
- const T & **operator[]** (size_t index) const
- **iterator begin** ()
- **iterator end** ()
- **const_iterator begin** () const
- **const_iterator end** () const
- template<typename P > void **rendez** (P pred)
- **~Vector** ()
  Destruktor.

Privát tagfüggvények

- **Vector** (const **Vector** &t)
- **Vector** & **operator=** (const **Vector** &t)

Privát attribútumok

- T * **pData**
  tárolt adatok tömbjének első elemére mutató pointer

- size_t **len**
  tárolt adatok száma

Részletes leírás

template<class T>

class Vector< T >

Változtatható méretű generikus tömb.

Paraméterek

T	- tárolt adattípus

Konstruktorok és destruktorok dokumentációja

template<class T > Vector< T >::Vector (const Vector< T > & t) [private]

template<class T > Vector< T >::Vector (size_t n = 0)

Default és olyan konstruktor egyben, amelyben meg lehet adni, hogy mekkorára inicializálja a vektort.

Paraméterek

n	- méret

template<class T > Vector< T >::~Vector

Destruktor.

Tagfüggvények dokumentációja

template<class T > iterator Vector< T >::begin () [inline]

Létrehoz egy iterátort és az elejére állítja.

Visszatérési érték

- iterátor az adatsorozat elejére

template<class T > const_iterator Vector< T >::begin () const [inline]

Létrehoz egy konstans objektumra alkalmazható iterátort és az elejére állítja.

Visszatérési érték

- iterátor az adatsorozat elejére

template<class T > iterator Vector< T >::end () [inline]

Létrehoz egy iterátort és az utolsó elem után állítja.

Visszatérési érték

- iterátor az adatsorozat végére

template<class T > const_iterator Vector< T >::end () const [inline]

Létrehoz egy iterátort és az utolsó elem után állítja.

Visszatérési érték

- iterátor az adatsorozat végére

template<class T > void Vector< T >::erase

Felszabadítja a vektor összes elemét.

template<class T > void Vector< T >::erase (iterator elem)

Kitöröl egy adott elemet a tömbből.

Paraméterek

elem	- iterátorral megadott elem, amit törölni szeretnénk

template<class T > Vector & Vector< T >::operator= (const Vector< T > & t) [private]

template<class T > T & Vector< T >::operator[] (size_t index)

Indexelő operátor.

Paraméterek

index	- index

Visszatérési érték

index. elem vagy std::out_of_range hiba

template<class T > const T & Vector< T >::operator[] (size_t index) const

Indexelő operátor konstans objektumra.

Paraméterek

index	- index

Visszatérési érték

index. elem vagy std::out_of_range hiba

template<class T > void Vector< T >::push_back (const T & data)

Egy új elem hozzáfűzése a vektor végéhez.

Paraméterek

data	- az adat, amit hozzá kell fűzni

template<class T > template<typename P > void Vector< T >::rendez (P pred) [inline]

Egy **Vector** típusú tároló rendezése egy adott predikátum szerint.

Paraméterek

vec	- rendezni kívánt vektor referenciája
pred	- predikátum, amelyet az összehasonlításnál használ az algoritmus

template<class T > size_t Vector< T >::size () const [inline]

Getter függvény a vektor méretére.

Visszatérési érték

méret

Adattagok dokumentációja

template<class T > size_t Vector< T >::len [private]

tárolt adatok száma

template<class T > T Vector< T >::pData [private]*

tárolt adatok tömbjének elsõ elemére mutató pointer

Vonat osztályreferencia

Publikus tagfüggvények

- **Vonat** (const **MyString** &**szam**, const **Datum** &**indulas**, const **Datum** &**erkezes**, int elsoAr, int masodAr, int elso=1, int masodik=0)
- **MyString getSzam** () const
- **Vector**< **Kocsi** * > & **getElsoOsztKocsik** ()
- **Vector**< **Kocsi** * > & **getMasodOsztKocsik** ()
- **Datum getIndulas** () const
- int **getElsoOsztAr** () const
- int **getMasodOsztAr** () const
- void **mentDatumok** (std::ostream &os) const
- virtual void **kiirVonat** (std::ostream &os) const =0
- virtual bool **megfelel** (const **MyString** &kezdo, const **MyString** &veg) const =0
- virtual bool **athalad** (const **MyString** &allomas) const =0
- virtual void **mentTipus** (std::ostream &os) const =0
- virtual void **mentAllomasok** (std::ostream &os) const =0

- virtual void **valtoztatAllomasok** (std::istream &is)=0
- virtual **MyString tipus** () const =0
- void **valtoztatIndulas** (const **Datum** &ind)
- void **valtoztatErkezes** (const **Datum** &erk)
- void **valtoztatElsoOsztAr** (int ujAr)
- void **valtoztatMasodOsztAr** (int ujAr)
- void **kiir** (std::ostream &os) const
- void **vasarolElsoOszt** (size_t &kocsi, size_t &hely)
- void **vasarolMasodOszt** (size_t &kocsi, size_t &hely)
- virtual **~Vonat** ()
  Virtuális destruktor.

Privát attribútumok

- **MyString szam**
  vonat azonosító száma

- **Vector< Kocsi * > elsoOsztKocsik**
  ***Vector** típus, amelyben dinamikusan foglalt első osztályú **Kocsi** objektumok vannak.*

- **Vector< Kocsi * > masodOsztKsocsik**
  ***Vector** típus, amelyben dinamikusan foglalt másodosztályú **Kocsi** objektumok vannak.*

- **Datum indulas**
  a vonat indulásának időpontja

- **Datum erkezes**
  a vonat érkezésének időpontja

- int **elsoOsztAr**
  egy első osztályú jegy ára a vonaton

- int **masodOsztAr**
  egy másodosztályú jegy ára a vonaton

Részletes leírás

Vonat osztály. Absztrakt osztály, az **Intercity** és Helyközi osztályok ősé. Egy vonattal kapcsolatos műveleteket képes elvégezni.

Konstruktorok és destruktorok dokumentációja

Vonat::Vonat (const MyString & szam, const Datum & indulas, const Datum & erkezes, int elsoAr, int masodAr, int elso = 1, int masodik = 0)

Konstruktor, amelyben meg kell adni minden adatot a vonattal kapcsolatban.

Paraméterek

szam	- vonat azonosító száma
indulas	- vonat indulásának időpontja

erkezes	- vonat érkezésének időpontja
elsoAr	- első osztályon egy jegy ára
masodAr	- másod osztályon egy jegy ára
elso	- első osztályú kocsik száma
masodik	- másod osztályú kocsik ára

Vonat::~Vonat () [virtual]

Virtuális destruktor.

Tagfüggvények dokumentációja

virtual bool Vonat::athalad (const MyString & allomas) const [pure virtual]

Eldönti, hogy a vonat áthalad-e vagy sem egy adott megállón.

Paraméterek

allomas	- megálló, amiről el kell dönteni, hogy az útvonal része-e

Visszatérési érték

igaz, ha a vonat áthalad a megadott megállón

int Vonat::getElsoOsztAr () const

Getter függvény, amely megadja, hogy az első osztályon mennyibe kerül egy jegy.

Visszatérési érték

első osztályú jegy ára

*Vector< Kocsi * > & Vonat::getElsoOsztKocsik ()*

A függvény egy referenciát ad vissza arra a **Vector** objektumra, amelyben az első osztályú kocsik vannak tárolva.

Visszatérési érték

referencia az első osztályú kocsikra

Datum Vonat::getIndulas () const

Getter függvény, amely visszaadja a vonat indulási időpontját.

Visszatérési érték

a vonat indulásának időpontja

int Vonat::getMasodOsztAr () const

Getter függvény, amely megadja, hogy az másodosztályon mennyibe kerül egy jegy.

Visszatérési érték

másodosztályú jegy ára

*Vector< Kocsi * > & Vonat::getMasodOsztKocsik ()*

A függvény egy referenciát ad vissza arra a **Vector** objektumra, amelyben az másodosztályú kocsik vannak tárolva.

Visszatérési érték

referencia az másodosztályú kocsikra

MyString Vonat::getSzam () const

Getter függvény, amely visszaadja az adott vonat azonosítószámát.

Visszatérési érték

a vonat azonosítószáma

void Vonat::kiir (std::ostream & os) const

A vonattal kapcsolatos általános adatok kiírása.

Paraméterek

os	- output stream ahova az adatokat ki szeretnénk írni

virtual void Vonat::kiirVonat (std::ostream & os) const [pure virtual]

Egy vonat kiírására szolgáló függvény.

Paraméterek

os	- output stream, ahova ki szeretnénk írni az adatokat

virtual bool Vonat::megfelel (const MyString & kezdo, const MyString & veg) const [pure virtual]

Eldönti, egy adott kezdő- és végállomásról, hogy benne van-e a vonat útvonalában.

Paraméterek

kezdo	- kezdő állomás
veg	- végállomás

Visszatérési érték

igaz, ha a vonat megfelel az útvonalnak

virtual void Vonat::mentAllomasok (std::ostream & os) const [pure virtual]

Elmenti az állomásait egy adott output streambe.

Paraméterek

os	- output stream ahova szeretnék menteni az állomásokat

void Vonat::mentDatumok (std::ostream & os) const

A függvény elmenti a vonat indulási, illetve érkezési időpontjait.

Paraméterek

os	- output stream arra a fájlra, amibe el akarjuk menteni a dátumokat

virtual void Vonat::mentTipus (std::ostream & os) const [pure virtual]

Elementi a vonat típusát.

Paraméterek

os	- output stream ahova a típust szeretnénk menteni

virtual MyString Vonat::tipus () const [pure virtual]

Egy MyStringet ad vissza, amiben a vonat típusa van.

Visszatérési érték

MyString típus, a vonat típusával

virtual void Vonat::valtoztatAllomasok (std::istream & is) [pure virtual]

A vonat állomásainak a megváltoztatására szolgáló függvény.

Paraméterek

is	- input stream, ahonnan jönnek az szükséges adatok

void Vonat::valtoztatElsoOsztAr (int ujAr)

A függvény megváltoztatja a vonat első osztályú jegyének árát.

Paraméterek

ujAr	- új első osztályú jegy ára

void Vonat::valtoztatErkezes (const Datum & erk)

A függvény megváltoztatja a vonat érkezési időpontját.

Paraméterek

erk	- az új érkezés időpontja

void Vonat::valtoztatIndulas (const Datum & ind)

A függvény megváltoztatja a vonat indulási időpontját.

Paraméterek

ind	- az új indulás időpontja

void Vonat::valtoztatMasodOsztAr (int ujAr)

A függvény megváltoztatja a vonat másodosztályú jegyének árát.

Paraméterek

ujAr	- új másodosztályú jegy ára

void Vonat::vasarolElsoOszt (size_t & kocsi, size_t & hely)

A függvény lefoglal egy első osztályú jegyet a vonaton, és a paramétereiben visszaadja a kocsi számát, illetve a hely sorszámát.

Paraméterek

kocsi	- referencia a lefoglalt kocsi sorszámára
hely	- referencia a lefoglalt hely sorszámára

void Vonat::vasarolMasodOszt (size_t & kocsi, size_t & hely)

A függvény lefoglal egy másodosztályú jegyet a vonaton, és a paramétereiben visszaadja a kocsi számát, illetve a hely sorszámát.

Paraméterek

kocsi	- referencia a lefoglalt kocsi sorszámára
hely	- referencia a lefoglalt hely sorszámára

Adattagok dokumentációja

int Vonat::elsoOsztAr [private]

egy elsõosztályú jegy ára a vonaton

*Vector<Kocsi *> Vonat::elsoOsztKocsik [private]*

Vector típus, amelyben dinamikusan foglalt elsõosztályú **Kocsi** objektumok vannak.

Datum Vonat::erkezes [private]

a vonat érkezésének idõpontja

Datum Vonat::indulas [private]

a vonat indulásának idõpontja

int Vonat::masodOsztAr [private]

egy másodosztályú jegy ára a vonaton

*Vector<Kocsi *> Vonat::masodOsztKsocsik [private]*

Vector típus, amelyben dinamikusan foglalt másodosztályú **Kocsi** objektumok vannak.

MyString Vonat::szam `[private]`

vonat azonosító száma

Függvények dokumentációja

datum.h fájlreferencia

Függvények

- std::ostream & **operator<<** (std::ostream &os, const **Datum** &d)
- std::istream & **operator>>** (std::istream &is, **Datum** &d)

Részletes leírás

Dátum osztály deklarációja és inline függvényei.

Függvények dokumentációja

std::ostream & operator<< (std::ostream & os, const Datum & d)

Globális inserter.

Paraméterek

os	- output stream referencia
s	- **Datum** referencia

Visszatérési érték

output stream referencia

std::istream & operator>> (std::istream & is, Datum & d)

Globális extractor.

Paraméterek

os	- input stream referencia
s	- **Datum** referencia

Visszatérési érték

input stream referencia

mav.h fájlreferencia

Függvények

- bool **vonatSzam** (const **Vonat** *a, const **Vonat** *b)
- bool **vonatDatum** (const **Vonat** *a, const **Vonat** *b)
- bool **jegyAr** (const **Jegy** &a, const **Jegy** &b)

Részletes leírás

Mav osztály deklarációja és inline függvényei.

Függvények dokumentációja

bool jegyAr (const Jegy & a, const Jegy & b)

Predikátum a jegyek ár szerinti rendezéséhez.

Paraméterek

a	- első vonat
b	- második vonat

Visszatérési érték

igaz ha az elsőnek kisebb a dátuma

*bool vonatDatum (const Vonat * a, const Vonat * b)*

Predikátum a vonatok dátum szerinti rendezéséhez.

Paraméterek

a	- első vonat
b	- második vonat

Visszatérési érték

igaz, ha az elsőnek kisebb a dátuma

*bool vonatSzam (const Vonat * a, const Vonat * b)*

Predikátum a vonatok szám szerinti rendezéséhez.

Paraméterek

a	- első vonat
b	- második vonat

Visszatérési érték

igaz, ha az elsőnek a száma előbbre van az abcben

menu.h fájlreferencia

Enumerációk

- enum **Allapotok** { **adatbazis**, **fomenu**, **kilep** }
  A program nézeteinek állapotai.

Részletes leírás

A program **Menu** osztályának deklarációja és inline függvényei.

Enumerációk dokumentációja

enum Allapotok

A program nézeteinek állapotai.

Enumeráció-értékek:

adatbazis	
fomenu	
kilep	

mystring.h fájlreferencia

Függvények

- std::ostream & **operator<<** (std::ostream &os, const **MyString** &s)
- std::istream & **operator>>** (std::istream &is, **MyString** &s)

Részletes leírás

MyString osztály deklarációja és inline függvényei.

Függvények dokumentációja

std::ostream & operator<< (std::ostream & os, const MyString & s)

Kiír az ostream-re.

Paraméterek

os	- ostream típusú objektum
s0	- String, amit kiírunk

Visszatérési érték

os

std::istream & operator>> (std::istream & is, MyString & s)

Beolvas az istream-rõl egy szót egy string-be.

Paraméterek

is	- istream típusú objektum
s0	- String, amibe beolvas

Visszatérési érték

is

teszteles.h fájlreferencia

Függvények

- void **tesztekFuttatasa** ()

Részletes leírás

Tesztelésre szánt függvények headerjeit tartalmazza

Függvények dokumentációja

void tesztekFuttatasa ()

Függvény, amely tartalmazza a teljes program teszteseteit. A tesztesetek a gtest_lite segítségével vannak megvalósítva.

Tesztprogram bemutatása

A *teszteles.h* fájlban lévő tesztprogram minden osztályt egyenként teszteli minden osztály publikus függvényeit. Esetleges hiba esetén, a Menu osztály *fociklus()* függvényében elkapja a hiba miatt dobott kivételt és ezt kezeli. A program három típusú kivételt kap el: std::bad_alloc, std::out_of_range és std::runtime_error. Az első két kivétel estén a program futását megállítja, az utóbbi esetén egy üzenetet ír ki a felhasználó számára, majd folytatódik a program futása. A fordítás során minden teszteset hibamentes, ami az interfész tesztnek, illetve funkcionális teszteseknek való megfelelését igazolja.

Memóriakezelés tesztje

A memóriakezelés ellenőrzését a laborgyakorlatokon használt MEMTRACE modullal végeztem. Ehhez minden önálló fordítási egységben include-oltam a ”memtrace.h” állományt a standard fejlécállományok után. Memóriakezelési hibát nem tapasztaltam a futtatások során.

Lefedettségi teszt

A funkcionális tesztek a program minden ágát lefedték.

A Cporta által meghatározott lefedettség 98,65%. Az a kis százalék, amely nincs lefedve, az az std::bad_alloc kivétel kezelése a Menu osztályban, illetve hibás indirekció a Vector osztályban. Ahhoz, hogy a tesztelés ezt is lefedje a *new* utasítás std::bad_alloc hibát kell dobjon.

Felhasználói kézikönyv

A program indításakor egy menü jelenik meg, amelynek tartalma a következő:

- **Meglévő adatbázis betöltése:** Ezt az opciót az 1-es billentyűvel választhatjuk ki. Ekkor a program kilistázza nekünk a már létrehozott adatbázisokat. Ezután a megfelelő sorszám begépelésével betölthetjük az adott adatbázist.
- **Új adatbázis:** Ezt a menüpontot a 2-es billentyűvel érheti el a felhasználó. Ekkor egy üres adatbázis jön létre, amelyben még semmi adat sincs elmentve.
- **Kilépés:** Ezt választva le lehet állítani a programot.

Mind a három fenti almenüből tovább lépve a főmenübe lép a program, ahol különböző lekérdezéseket lehet megjeleníteni és módosításokat lehet végrehajtani az aktuális adatbázissal kapcsolatban. Ezek az opciók a következők:

16. **Vonat hozzáadása [1]:** Először a vonat típusát kell megadni majd a vonat egyedi számát. Ezután a program sorban fogja kérni, hogy hány első osztályú, másod osztályú kocsija van a szerelvénynek, illetve az osztályonkénti árat. Ezt követően meg kell adni az indulás és érkezés dátumát és időpontját. Majd ezek után intercity vonat esetén a kezdő állomást és végállomást. Helyközi vonat esetén azt kell megadni, hogy hány megállója van a szerelvénynek, és az állomások neveit egyenként.

17. **Jegyvásárlás [2]:** Ehhez meg kell adni. hogy honnan indulunk és hová szeretnénk érkezni. A program megnézi, hogy van-e megfelelő vonat és kilistázza őket. Ha nincs megfelelő vonat, akkor ezt jelzi egy üzenet formájában. A kilistázott vonatok listájában az is szerepel, hogy az adott vonat milyen típusú, mikor indul és érkezik a megadott helyiségbe, illetve, hogy mennyibe kerül egy jegy első és másodosztályon. A felhasználónak meg kell adni a vonat

számát. Ezek után a program egy megerősítést ír, amelyben fel lesz tüntetve a kocsi száma, illetve a lefoglalt hely.

18. **Vonat törlése [3]:** Kilistázódnak a vonatok, majd meg kell adni, hogy melyik vonatot szeretnénk törölni. A vonat törlésével az adott vonatra vonatkozó jegyek is törlődnek.

19. **Jegy törlése [4]:** Kilistázódnak a jegyek, ezután a megfelelő sorszámot megadva törölni lehet a jegyet. Ekkor az adott hely újra üres lesz a vonaton.

20. **Útvonal megváltoztatása [5]:** Intercity vonat esetén meg kell adni az új indulás és érkezési állomást. Helyközi vonat esetén meg kell adni, hogy hány megállója van az új útvonalnak, majd az állomásokat. Ami nagyon fontos, hogy annak a vonatnak, amelynek megváltoztatjuk az útvonalát, annak törlődnek a jegyei is, mert ezek már nem aktuálisak.

21. **Menetrend megváltoztatása [6]:** Ki kell választani, hogy melyik vonat menetrendjét szeretnénk megváltoztatni, majd meg kell adni sorra az új indulás, illetve érkezés időpontját.

22. **Árak megváltoztatása [7]:** Miután kiválasztunk egy vonatot, kiválasztjuk, hogy melyik osztálynak szeretnénk megváltoztatni az árát és megadjuk az új árat.

23. **Vonatok rendezése szám szerint [8]:** Ezt az opciót választva a program rendezi a már felvett vonatokat szám szerint, és újra kilistázza őket.

24. **Vonatok rendezése dátum szerint [9]:** Ezt az opciót választva a program rendezi a már felvett vonatokat indulási időpontjuk szerint, és újra kilistázza őket.

25. **Vonatok szűrése típus szerint [10]:** Meg kell adnunk, hogy milyen típusú (intercity, helyközi) vonatokat szeretnénk kilistázni, majd csak a választott típusúak jelennek meg.

26. **Vonatok szűrése állomás szerint [11]:** Meg kell adnunk egy állomás nevét, majd azok a vonatok listázódnak ki, amelyek ezen az állomáson keresztülhaladnak.

27. **Összes vonat listázása [12]**

28. **Jegyek rendezése áruk szerint [13]:** Rendezi a jegyeket áruk szerinti növekvő sorrendben, majd kilistázza őket.

29. **Összes jegy listázása [14]**

30. **Mentés [15]:** Ha elvégeztük az általunk kívánt módosításokat az adatbázison akkor ezt az opciót választva el lehet menteni a változtatásokat. Abban az esetben, ha előzőekben új adatbázist hoztunk létre, akkor mentés előtt a program fog kérni egy nevet, ami segítségével azonosítani tudjuk majd az adatbázist. Ennek a névnek nem szabad megegyeznie az eddig elmentett adatbázisok neveivel. Ha ez mégis megtörténik ezt a program jelezni fogja egy üzenet formájában. Mentés után újra az első menü fog megjelenni.