

# Vonatjegy Specifikáció

## Feladat

A feladat egy vonatjegyeket kezelő rendszer létrehozása. Alapvetően két feladatot kell ellátnia a rendszernek: vonatok járatok felvétele és vonatjegyek kiadása.

Egy jegynek tartalmaznia kell: *A vonatszámot, kocsiszámot, hely számát, indulás és érkező állomás nevét és indulás, illetve érkezés idejét.*

Minden kiadott jegyhez helyjegyet is kötelezőek vagyunk adni, így helyjegy nélküli jegyet nem kell kezelni.

## Specifikáció

A program maga konzolos felületen fog működni. Az eltárolandó adatokat fájlból fogja kiolvasni, illetve, ahogy a feladat kéri fájlba írás is szükséges. Ezen felül mivel a rendszer nem véges (*nem tudjuk előre mennyi vonat vagy kocsis lesz*), ezért dinamikusan foglalt memóriaterületen fog dolgozni.

A program indításakor egy menürendszeren keresztül lehet majd navigálni a funkciók között:

```
Udvozlom a vonatjegyeket kezelo programban!  
Válasszon menupontot:  
(1) Vonat hozzaadasa  
(2) Vonatjegy vasarlasa  
(3) Vonatok listazasa  
(42) Kilepes
```

Ha vonatot szeretnénk felvenni, akkor egy txt fájlba fogja a program a vonat adatait kiírni. Vonatjegy kiadása esetén a program ebből a txt fájlból fogja az adatokat kiolvasni.

Jegyvásárlás esetén egy kereső rendszer jelenik meg. Keresés során a megadott értékek alapján (pl: Induló állomás, indulás ideje, érkező állomás) kilistázza a paramétereknek megfelelő vonatokat, majd az általunk kiválasztott vonat azonosító számkódját beírva véglegesítjük a vásárlást. A hibamentes jegykiadás után (nem volt tele a vonat stb...) a program jelzi, hogy sikeres volt a folyamat és a jegy adatait kiírja egy txt fájlba.

A vonatok listázása funkció kilistázza az összes elérhető vonatot a rendszerben megjelenítve a járat adatait illetve a szabad helyek számát.

*Később akár például törlés funkció is bekerülhet a menübe.*

## OOP

A program hatékony működés, illetve bővíthetőség érdekében szükség van a programban osztályokra és azok helyes meghatározására.

Előre láthatólag ezekre a class-okra lesz szükségünk:

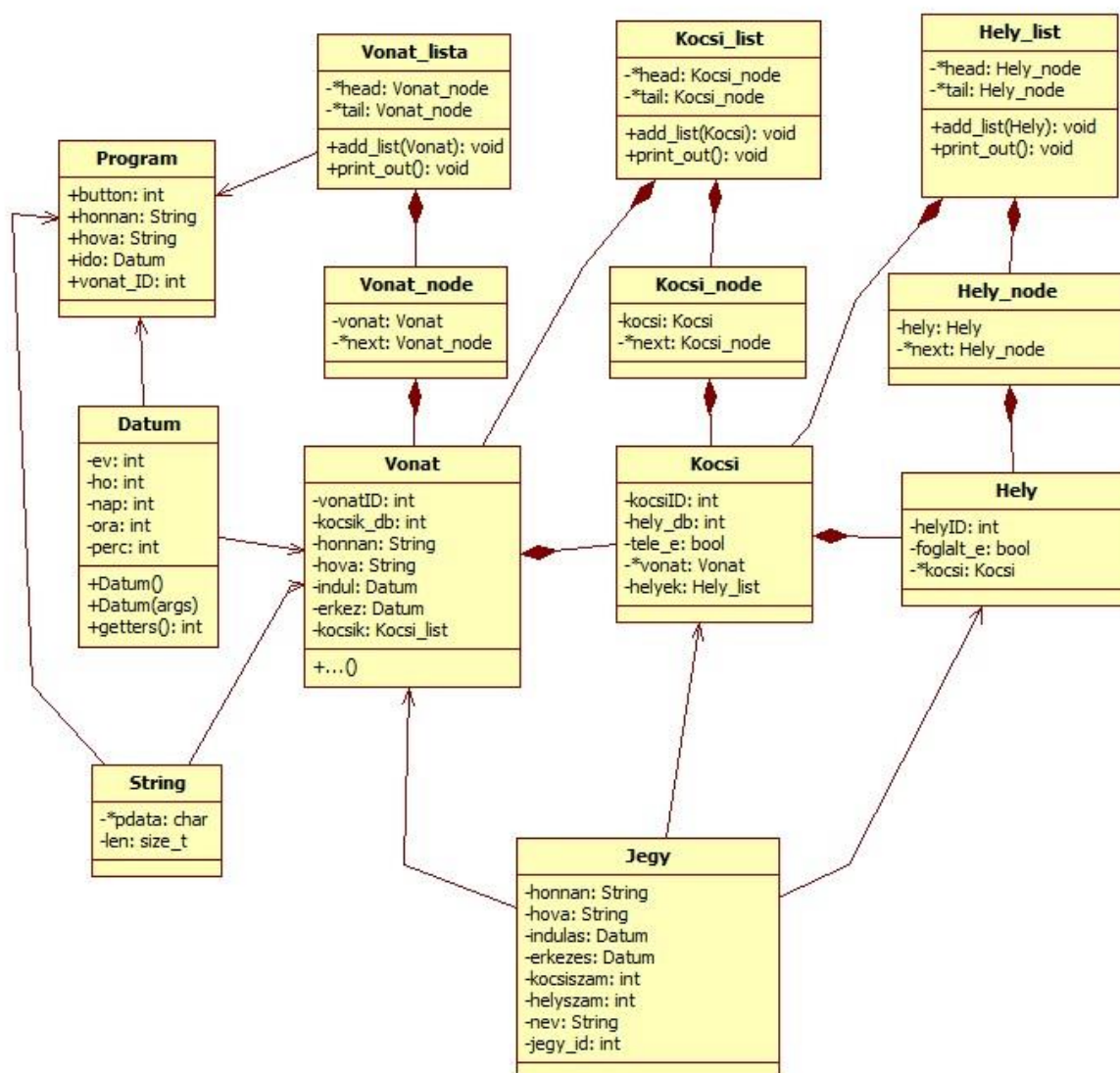
- **Vonat class**  
Itt tárolhatjuk, hogy honnan, hova megyünk. Indulás és érkezés időpontját. Kocsikra mutató pointereket.
- **Kocsi class**  
Kocsiszám tárolása, a helyek tömbje.
- **Hely class**  
A hely száma, illetve egy bool érték, hogy foglalt-e már vagy sem.

A későbbiekben a jegynek is lehet class-ja. A bővítés lehetősége nyitott.

## Terv

## Az objektum modell:

A programban felhasználok a laborfeladatokon elkészített String osztályt és egy módosított Datum osztályt. Ezeket csak a Vonat illetve a Program osztályban fogom felhasználni.



A Vonat-Kocsi-Hely osztály hármas tagfüggvényei konstruktorok, destruktorok, tagfüggvények getterei, illetve ostreamre való kiírás.

A Vonat osztály tartalmazza a Kocsi\_list-t ami a kocsik láncolt listája. Ez a lista bejárható és kiíratható. Ugyanilyen kapcsolata van a Kocsi és Hely list-nek.

A Vonat\_list szintén láncolt listában a vonatokat tartalmazza. Ez szintén kiíratható, ennek a feladat megoldásának szempontjából a listázásnál lesz szerepe, hogy meg tudjuk jeleníteni az összes elérhető vonatot.

Amikor egy Jegy-et létrehozunk, akkor az hivatkozni fog a Vonat-Kocsi-Hely számára lényeges tartalmára (azaz: honnan, hova, időpontok, vonatszám, kocsiszám, helyszám).

A program vezérléséért a program osztály fog felelni. Itt fog történni a menü kezelés illetve a fájl műveletek is.

## Algoritmusok:

Vonat hozzáadásakor a paraméteres konstruktor hívódik meg. Ekkor létrejön egy üres Kocsi\_list is, majd egy ciklusban, a kocsik számától függően, meghívódnak a kocsik konstruktorai, amik szintén meghívják a helyek konstruktorait és minden meghívott objektumot hozzáfűz a program a számára megfelelő listába.

Jegy vételekor, végig megy a láncolt lista elemein, amíg nem talál egy szabad kocsit, majd szabad helyet. Amikor a találat sikeres, az adott helyen a foglalt\_e adattagját átállítja foglaltra (ha a kocsi is megtelik annak is). Jegy törlése esetén megkeresi az adott helyet és visszaállítja a bool adattagot.

## Megvalósítás

A feladat megoldása során az eredeti tervhez képest változások keletkeztek.

A jegy osztályt teljesen kihagytam. Nem volt rá szükség, inkább a vezérlő osztállyal, illetve néhány segéd függvénnyel a fő osztályokon belül megoldottam a jegy kezelésének problémáit.

A vonat-kocsi-hely listáknak elkészítettem egy sablon láncolt listát, így szintén sablon lett a Node struktúra is.

A vonat-kocsi-hely osztályok paramétereiben nem a listát kapják, hanem az első lista elemét. Erre a listák átfedése miatt létrejött adatvesztés miatt volt szükség. A program osztály is csak az lista első elemét kapta meg.

A programnak egyetlen egy paramétere lett, az pedig a vonatLista\_head.

A string5.h és string5.cpp fájlokat teljesen elhagytam a programból, mert inkább a std::string -et alkalmaztam helyette.

A vezérlő programban elkészített függvények után, csak a mainben a test függvény megvalósítása kellett. Ehhez a gtest\_lite.h -t alkalmaztam.

## A template lista:

A template lista template elemeket tartalmaz. Egyszeresen láncolt, csak egy irányból járható be.

A template Node tartalmaz:

- Pointert az adattagra
- Pointert a következő elemre

A láncolt lista paramétere a head template elem, ami az első elemet tartalmazza, mindig minden körülmények között.

Tagfüggvényei a következők:

- default konstruktor /head = NULL/
- másoló konstruktor
- head getter
- operator =
- add\_node(T \*data) /A láncolt listába fűz egy elemet/

## A vezérlő program:

A vezérlő program csak a vonatok láncolt listáját tartalmazza. A vezérlést teljes mértékűleg ő végzi el. Vele indítjuk a programot és vele is kell bezárni. Erre vannak tagfüggvények.

### Függvényei:

- **default konstruktor**  
/vonatLista\_head = NULL/
- **void program\_start()**  
/Elindítja a programot, egy switch-case menüvel. Tartalmazza az összes tagfüggvényt./
- **void vonat\_felvesz()**  
/A vonat felvételére szolgáló függvény. Beleírja a vonat.txt fileba a vonatot, majd meghívja a file\_beolvas() függvényt./
- **void file\_beolvas()**  
/A fileből kiolvassa soronként az adatokat, meghívja a vonat konstruktort majd ezek után befűzi őket a listába és meghívja a foglalt\_check függvényt./

- ***void foglalt\_check***  
/A foglalt.txt fájlból kiolvassa a foglalt helyeket, és a listában végigmegy a vonatokon, kocsikon és helyeken és átállítja azokat foglaltra./
- ***void listaz\_ossz()***  
/Az összes vonatot kilistázza/
- ***void vonat\_keres(string honnan="", string hova="", Datum ido = Datum())***  
/Vonatot lehet keresni vele három féle képen. Csak induló állomás megadásával, induló és érkező állomás megadásával, induló, érkező és dátum megadásával. (Gyakorlatilag 4 módon, mert ha a felhasználó nem ad meg semmilyen paramétert, akkor minden vonatot ami elérhető kilistázik.)/
- ***void vonat\_torol(int vonatID)***  
/Kitörli az adott azonosítóval rendelkező vonatot a vonat.txt fájlból. Meghívja a vonat\_torol\_foglalt függvényt./
- ***void vonat\_torol\_foglalt(int vonatID)***  
/Törli a foglalt.txt fájlból az adott azonosítójú vonatra szóló foglalásokat/
- ***void jegy\_vetel(int vonatID, int kocsi = 0, int hely = 0)***  
/Jegyet vesz, beleírja a foglalt.txt-be, ha sikeres a jegy vétel. Létrehoz a jegy\_name és jegy\_txt segítségével egy txt fájlt a jegynek./
- ***void program\_exit()***  
/Felszabadítja az összes foglalt memóriát, majd bezárja magát/

## A tesztelés

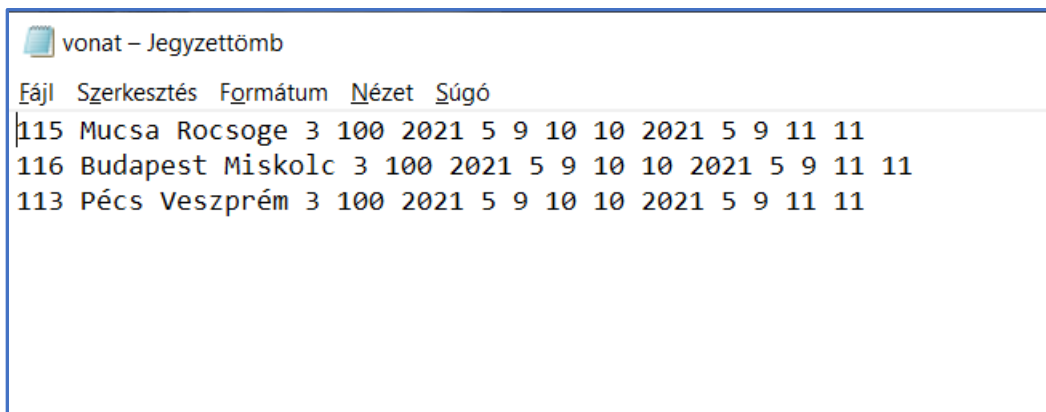
A program tesztelésére egy függvényt hoztam létre, ami a gtest\_lite.h -t használja fel segítségül. A kilenc tesztetben tesztelem a **Hely - Kocsi - Vonat** osztályok konstruktorait, gettereit, settereit, illetve ezáltal a sablon lista működését is. Ezen kívül a Vonat osztály tagfüggvényeiből a hely\_switch és hely\_check függvényeket, ez a két függvény, amit a vezérlő program felhasznál.

A vezérlő programot nem teszteli, mivel a vezérlő rész csak az imént tesztelt szubrutinokat használja fel.

## A forrás fájlok

A programban 2 forrásfájl található: a vonat.txt, illetve a foglalt.txt.

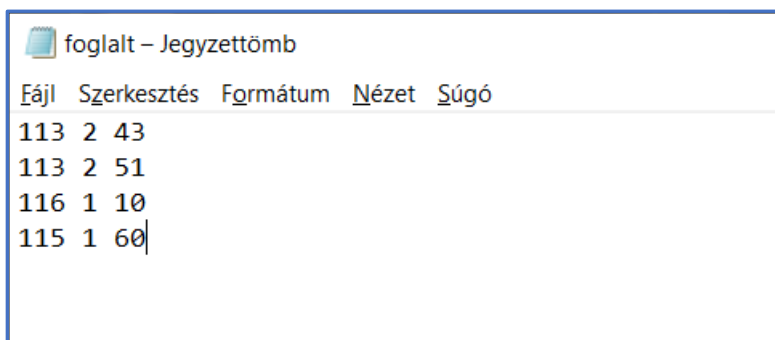
vonat.txt felépítése:



115	Mucsa	Rocsoqe	3	100	2021	5	9	10	10	2021	5	9	11	11
116	Budapest	Miskolc	3	100	2021	5	9	10	10	2021	5	9	11	11
113	Pécs	Veszprém	3	100	2021	5	9	10	10	2021	5	9	11	11

VonatID - Induló állomás - Érkező állomás - Kocsi db - Hely db/kocsi - induló dátum - érkező dátum

foglalt.txt felépítése:



113	2	43
113	2	51
116	1	10
115	1	60

VonatID - KocsiID - HelyID