# Házi Feladat

Készítette: Mikáczó János

Nagy Házi Specifikáció

## Feladat:

## Vonatjegy

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!

## Specifikáció:

Vonatjegy eladó automata:

-vonatok és menetrendek: Fájlból olvasás és fájlba írás, ebben az állományban lévő adattagok módosíthatóak a programmal: Kocsik száma, és maximum férőhely definiálása, indulási, érkezési hely és idő módosítsa.

-Jegyrendszer: Legyen átszállási lehetőség, kell egy összeadó operátor, amivel meg lehet oldani az átszállást, vagy pedig túlterhelve több jegyet venni. Figyelembe kell venni a megegyező állomásokat, és a szabályrendszert ami alapján lehetséges ezeket megtenni.

-A helyjegy megszűnik, újra kioszthető.

Kimenet, bemenet: Konzolos applikáció, ami egy menürendszer formájában kiválasztós, illetve input mezők szerint vezérelhető, így lehet az adatokat felvenni, törölni stb….

A jegyeket lehet „nyomtatni” azaz fájlba menteni, ezeket többszörösen tárolni, így pl.: egy „ember számára jobban kinéző” jegyet megalkotni, amin van egy azonosító kód ami egy másik fájlban meghatároz egy a program számára kedvezőbb formátumban beolvasható sort, így az lesz a fő állomány, majd ebben lehet a jegyeket módosítani, és újra kinyomtatni.

# Terv:

Vonatjegy eladó automata:

Nincs ár, a feladatban nem releváns egy árkalkuláció.
Fájlból olvasás: Egy osztályból példányosított dinamikus tömbbe kerülnek a járatok, Ezen belül mindegyik járatnak lesz még 2 komplexebb adattagja, az egyik a megállók ami a laboron megoldott String osztály segítségével lesz tárolva, a másik egy időpont, így lekérdezhető lesz egy getter segítségével és egy index számmal lekérdezni, hogy az a járat n-edik megállóba mikor fog eljutni.

Szinte minden dinamikus memóriában foglal helyet, így nem lesz maximuma a legtöbb tárolóegységnek.

Minden járaton van egy vonat ami kocsikból áll, a kocsikban van hely, ami elfogyhat, így minden kocsi felvételekor vagy kitörlésekor egy fix mennyiségű hellyel fog változni.

Úgy tervezem, hogy a megállókat utólag törölni nem lehet, így nem kell mindig rendezni a tömböt, így csak felvenni lehet a végére, ezzel tolódna az idő és a sorrend is.

Jegyrendszer:

Jegyvásárlásnál, egy keresést fogok használni arra, hogy megtalálja a program a keresett járatot.

Ez úgy működik, hogy mikor az utas átszállással akar utazni, akkor megteheti manuálisan is, hogy úgy vesz jegyet, de automatikusan ilyenkor a program ha nem egy járatot talál, hanem kettőt akkor megkeresi hogy melyik járatokkal lehet majd eljutni, majd ha ez megvan akkor néz egy időpontot és emiatt fontos tárolni mindegyik megállónak az érkezési idejét, mert így egy utána leghamarabb induló járatot tud majd választani.

A jegy helyjegy is, így ez kiosztásra kerül, ekkor a megállóknál külön szereplő helyszám egyel csökken, amik érintettek, ez egy érdekes megvalósítás, és ez tűnik a legkönnyebben megvalósíthatónak, így aki pl.: Debrecentől Szolnokig vesz jegyek, akkor Szolnok után már az elérhető helyek száma nem változik

Egy egyszerű elágazással ez ellenőrizhető és így a jegyvásárlás félbeszakad.

Kimenet/bemenet:

A kimenet egy konzolra kerülő menü lesz, nagyon egyszerűen 1-x ig számok közül lehet majd válogatni, lesznek olyan menüpontok amik utána lehet gépeléssel megadni adatokat, pl a Jegyeknél a név mező.

Fájlból olvasás: a fájlok egy speciális karakterrel elválasztott soronként egy bejegyzéssel lesznek tárolva, így nem egy bonyolult feladat beolvasni ezeket. Erre lesz egy külön függvény pl *menetrend.beolvas(„fájlnév”)*

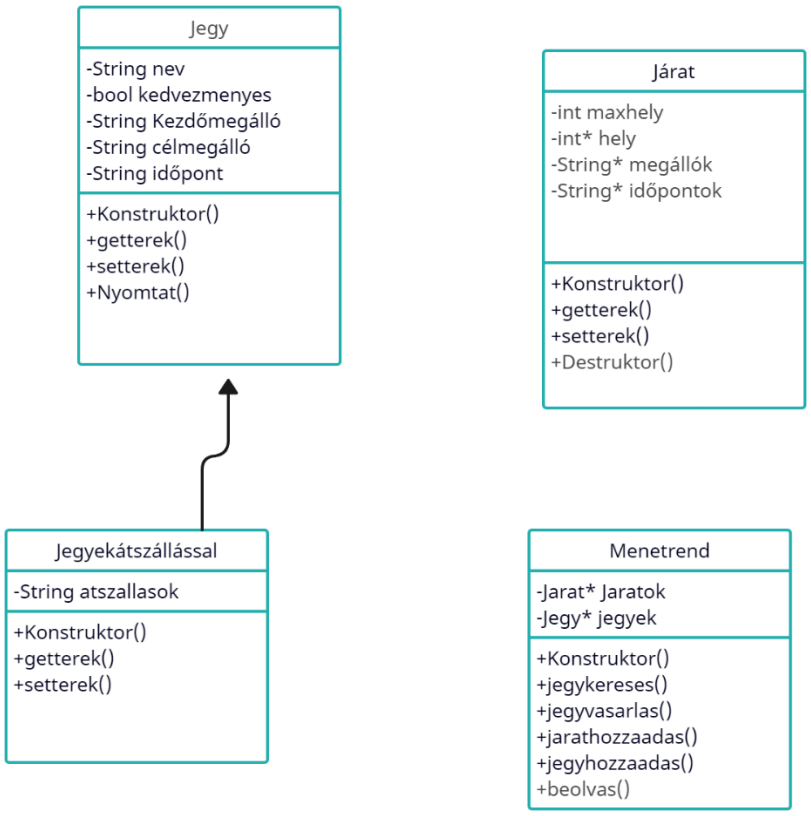
jegyek nyomtatása egy azonosítóval kezdődik ami az első sor, így csak addig kell majd beolvasni, ezen szerepel majd:

#####azonosító#####
Név:
Típus: kedvezményes/nélküli
Mikor:
Mettől- meddig:
helyszám:
######################

Kereső algoritmus váza:

Bekér: kezdő és végállomás
*Keres(){*
*Végig lépkedünk a járatokon:*
*Mindegyik járat, mindegyik megállóját végig nézzük, és ha szerepel mindkét megálló akkor találtunk egy megfelelő járatot*
*Máskülönben:*
*Megkeressük a kezdő megállót, majd nézünk Közös megállókat a többivel, ha van hozzáadjuk egy listához*
*Ciklus vége*
*ha nincs találat akkor sikertelen a keresés, ha van akkor:*
*Végig lépkedünk a közös megállókon, majd mindegyikre meghívjuk a keres függvényt a közös és a cél megállóra. Ha van akkor meg lesz találva.*
*Ciklus vége}*

**UML DIAGRAMM:**




Források: UML diagramm szerkesztő: https://app.creately.com/ (2024.04.21)

# Megvalósítás:

A feladat, nagyban hasonlít a terve, négy darab osztályból, és egy felhasználói felületből áll.
Nem használ semmilyen includeolt könyvtárat az iostream és fstream kivételével.
A függvényeknél törekedtem arra, hogy csak azokat valósítsam meg amit használok is, így sok setter feleslegessé vált a későbbiekben, a kereső algoritmus módosult.

Módosítások: a jegyek keresésére a bevezetés a számelméletbe 2 tárgyból a BFS algoritmus ihletett meg és kivitelezhetőnek tartottam, átszállást csak 1 darab átszállás esetén tud megvalósítani.
A jegyeket nem lehet fájlból olvasni, elsőnek jó ötletnek tűnt, de rájöttem, hogy felesleges, hiszen a jegyeket nem betölteni akarjuk, hanem kiadni. A járatok kiírhatóak és beolvashatóak, illetve a jegyek is kiírhatóak fájlba is. Járatok felvehetőek, manuálisan átszállás nem oldható meg, az csak egy újabb input mező lett volna.

Osztályok bemutatása:

## Járat osztály:

UML diagrammban meghatározott adattagokon kívül bővült egy int megallokszama-val.

Függvények tekintetében ami nem egyértelmű teljesen az a jaratkiir(), és a jaratkiir_fajlba(), a nevük árulkodó viszont, és innen következik a különbség is.

jegyfoglal() függvény: lefoglalja a jegyet, azaz pontosabban a helyet, egy adott indexű megállóban, ezt később a komplexebb függvényben alkalmaztam ami a jegyteszt függvény volt.

## Jegy osztály:

Adattagok nem bővültek, annyi a módosítás, hogy az időpont int típusú.

Nyomtat() függvényen kívül ami nem az alapokhoz tartozik az egy operator= és egy jegykiirfajlba() függvény.

## Jegyatszallas osztály:

A Jegy osztály a szülő osztálya, és virtuális függvények miatt 2 szinte semmi más változás nincs benne.

## Menetrend osztály:

Ez tartalmazza az előző három osztályt, és azokból is többet, tárol még egy jegyek számát, járatok számát, és az átszállásos jegyek számát, de ez nem olyan érdekes még.

Itt valósul meg a hozzáadó függvények amik csak felvesznek egy újabb tagot, mindegyik elem dinamikusan van foglalva így tetszőlegesen sok elem felvehető, itt valósul meg a menetrend betöltése is, illetve a jegy keresése

Van egy globális függvény a jegyteszt.

A legtöbb konstruktornak van default és paraméterezhető verziója is megvalósítva, és használva is.

Szinte az összes függvény a vonatok.cpp-ben van megvalósítva, kivéve néhány amit .h-ba írtam mert olyan szinten rövidnek bizonyult, így például egy getter-t nem írtam külön meg.

Hogyan épül fel maga a program?

a main.cpp-ben van egy menü, amibe van egy kikommentezett rész, ez egy „hidegindító”-ként szolgál, ha nincs meg a fájl amiben a járatok vannak, ekkor létrehozza és hozzáadja őket a menetrendekhez. Egy menü sor jelenik meg amiben lehet választani 1.-től 8.-ig. Az első kettő pont amit lehet választani azt amiatt választottam annak, mert ezek azok amikre való a program, a felhasználó tud jegyet venni és keresni egy járatot a menetrendben, nyilván előtte be kell tölteni,

csak számot fogad el, és egyéb karakterre újra bekér egy adato. Ez Switch case-el lett megolva.

Menjünk sorba:

1.Jegyvásárlás
Bekér egy nevet, kezdőmegállót, célállomást. ide 100 méretű tömböket használtam, hisz nem gondolnám, hogy ez akkora nagy baj lenne, ezt megkapja egy Jegy, aminek a konstruktor paramétereibe beilleszti a kapott adatokat.

Meghívódik a jegyteszt:

A jegyteszt bekér egy menetrendet és egy jegyet mint referencia érték, majd a jegy adataival keres egy útvonalat a jegykeres függvény segítségével, ha talált akkor lefoglalja a helyet, majd hozzáadja a menetrendhez, de ennél sokkal több minden történik, itt dől el, hogy sima vagy átszállásos lesz a jegy, hogy az időpont amit talál mi is lesz, illetve az átszállási pont megkeresése is itt történik.

2. Jegy keresés:
Ez csak olyan mint egy menetrend keresés, hasonló az első ponthoz csak nem ad hozzá jegyet csak a jegykeresés függvényt teszteli felhasználói bemenetekkel.

3. Kilépés
Kilép a programból, ezzel érdemes megtenni mert ekkor fut le az összes destruktor.

4.Mentés
Lementi fájlba a járatokat, azok maximális és aktuális helyeit illetve időpontjait. Nem a felhasználótól kér be fájlnevet, hanem egy jaratok fajlba kimenti és ennyi.

5.Betöltés
Ez egy fontos pont lehet egy felhasználónak mivel ha elindítja a programot és nincs betöltve a menetrend akkor nem fog tudni jegyet vásárolni illetve keresni. Ez a kikommentelt szekció feloldásával visszaállítható és így tudunk tesztelni könnyebben.

6.Teszt
Egy gyors teszt fut le ami egybefoglalja az előző lépéseket, ezt használtam folyamatosan hogy megtudjam minden működik-e. Betölt egy menetrendet, majd hozzáad 4 jegyet, illetve egy járatot manuálisan. Mindegyik jegy különböző fajta tesztesetek, nincs hely, nem található megálló, van megálló és átszállásos megálló esetet valósít meg, majd kiírja a menetrendet, a jegyeket és azok útvonalát ha van, illetve fájlba is menti a jegyeket.

7.Járat hozzáadása
Csak szimplán bekér egy maximális hely számot, megállók számát, megállók neveit majd hozzáadja a fő menetrendhez ezt a járatot.

8.Menetrtend kilistázása
Kiírja az összes járatot.

Ha a felhasználó negatív számot akar beírni időpontnak akkor a program jelez, illetve a fájlműveleteknél is van elhelyezve kivételkezelés

Memóriakezelés:
nem használtam memtrace.h-t, valamilyen oknál fogva nem működött nálam csak nagyon specifikus esetekben, így valgrindot használtam, ami tökéletesen működött.

Létezik járatszám, és ez alapján kap mindegyik járat egy azonosítót, ezt nem írom ki a jegyre, mert csak keresek vele, a helyszám nem működik, viszont megállónként leveszi a helyet, és egy idő után kiírja, hogy nincs hely…  ---„ez volt legutoljára megvalósítva”---

Bővíthető az egész program, például több átszállásra, illetve képes lenne idő szerint is keresni, csak akkor tömböt kéne átadni amiben benne van az összes járat ami indulási idővel egyenértékű vagy nagyobb, illetve a megállók helyesek.