# HÁZI FELADAT

Programozás alapjai 2.

Dokumentáció

Glocker ENdre
KBC838

2022. május 14.

**Programozói dokumentáció**

**Fordítási környezet:**

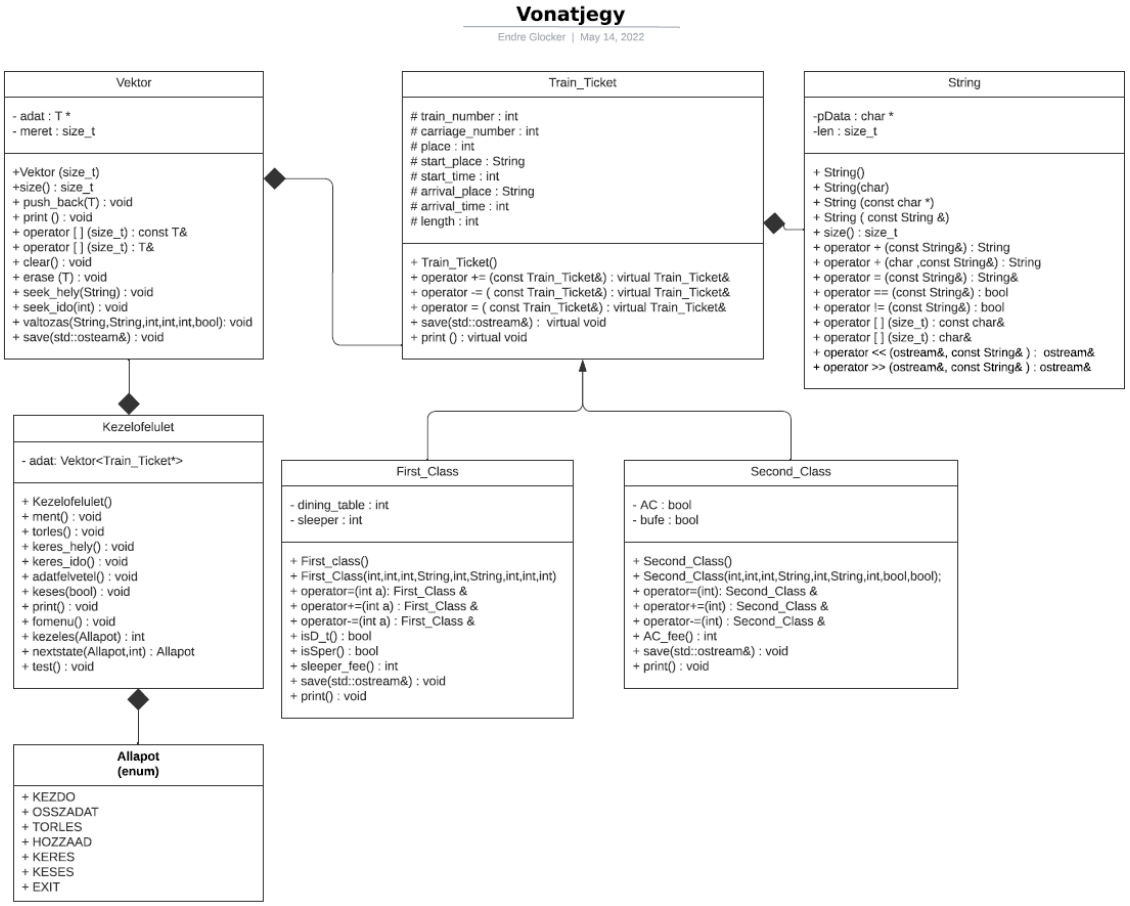
1. A vonatjegy.exe program futtatásához szükséges a MinGW GCC compiler
2. Szabványos függvénykönyvtárok és gtest_lite

**Fordítás folyamata:**

1. Létre kell hozni a "kezelofelulet.o", " stringek.o", "tickets.o", "memtrace.o", "main.o" nevű fájlokat a parancssorban: gcc -c file_nev.c -o file_nev.o
2. Létre kell hozni a konyvtar.exe fájlt: gcc f ilementes.o fileolvasas.o … main.o -o vonatjegy.exe

**Vonatjegy:**

A program célja egy vonatjegy adatbázis felállítása objektum orientált elemek használatával. A program képes a menetidők megváltoztatására, elemek törlésére és elemek felvételére az adatbázisba. Emellett még képes keresést is végrehajtani benne helyszín vagy idő alapján.




**Program működését vezérlő modulok**:

1. **String (** stringek.cpp & stringek.h):

   A String osztály felelős a feladatban levő karakterláncok / sztringek kezeléséért

2. **Tickets (** tickets.cpp & tickets.h):

   A programhoz tartozó fő adatok és az azokkal kapcsolatos műveletek itt vannak eltárólva.

3. **Vektor** (vektorok.h):
   A programhoz tartozó adatok fő tárolószerkezete itt van megvalósítva
4. **Kezelofelulet** ( kezelofelulet.cpp & kezelofelulet.h):
   Itt megy végbe a fájlok beolvasása és mentése, itt található a program főmenüje és az ahhoz tartozó tevékenységek, emellett a tesztesetek is itt vannak megvalósítva.

**A program működését vezérlő osztályok:**

**String osztály:**

**Privát adattagok:**

**char *pData :** a pData adattag tárolja a String változók karakterláncolatait

**size_t len :** eltárolja a sztringek hosszát

**Publikus függvények:**

**String() :** A paraméter nélküli hívható konstruktor a beállítja a *pData* értékét NULL-ra és a *len*-t pedig 0-ra.

**~String():** Felszabadítja a lefoglalt memóriaterületeket, amikor az objektum megszűnik.

**String(char) :** A konstruktor paraméterként kap egy karakter típusú változót, majd lefoglal egy 1 karakternyi memóriaterületet a *pData* változónak emellett a *len*-t 1-re állítja.

**String(const char* ):** A konstruktor paraméterül kap egy karakterláncot, a konstruktor beállítja a *len*-t a kapott paraméter hosszára, majd a karaktereket belemásolja a *pData*-ba.

**String(const String& ):** A másoló konstruktór paraméterül kap egy *String*-et, majd *len*-t beállítja annak a hosszára és *pDatá*ba belemásolja a paraméterként kapott *String pDatá*ját.

**get_pData():** A függvény az osztályon kívüli kommunikációt biztosítja (getter), visszatérési értéke egy *karakter mutató*, *pDatá*val tér vissza a függvény.

**size():** A függvény az osztályon kívüli kommunikációt biztosítja (getter), visszatérési értéke *size_t* és visszaadja az adott *String* hosszát.

**iras():** Az irás függvény a *String*-ek normál bemenetről és kimenetre történő olvasásában és írásában ( **operator**<< és **operator**>>) vesz részt. A függvény visszatér a *pData* karakter pointerrel ha *len* nagyobb mint nulla, ha *len* nulla, akkor üres karakterrel (" ") tér vissza.

**String& operator = (const String &):** Az i String&

**operator = = (const String &):** Az operátor összehasonlítja a balértékű *String*et a jobbértékű *String*gel, ha azok megegyeznek akkor igaz értékkel tér vissza, különböznek hamis értékkel tér vissza. A függvény először megnézi, hogy a balérték *String*jének hossza megegyezik a jobbérték *String* hosszával, ha nem, akkor már biztosan nem egyezi meg a két *String*, ha a méretek egyenlőek, akkor a függvény karakterenként összehasonlítja a kapott *String*eket. Ha minden karaktere egyenlő, akkor a függvény igaz értékkel tér vissza.

**operator ! = (const String &):** A nemegyenlő operátor balértékként kap egy Stringet, amit összehasonlít a jobbérték Stringgel ha nem egyenlő a két oldal visszatér igaz értékkel, ha a két oldal egyenlő akkor hamis értéket ad. Megvalósítás: a függvény visszatérési értéke az egyenlőség operátor függvényének negáltja ( **return !(*this = = rhs);** ).

**operator +( const String &):** Az operátor jobbértékként kap egy *String* változót, amit hozzáfűz az eredeti *String*hez az alábbi módon: létrehoz egy átmeneti *String*et, aminek a hosszát egyenlővé teszi a balérték és jobbérték hosszának az összegével (temp.len = len + rhs.len) majd lefoglal az átmenetinek egy az összegnél eggyel nagyobb memória területet (a lezáró nulla miatt kell a +1), az átmeneti változó *pDatá*jának, majd belemásólja a balérték *pDatá*ját utána a jobbérték *pDatá*ját. Végül visszatér az átmeneti változóval.

**operator [](size_t):** Az indexelő operátor jobbértékként kap egy size_t változót, ha a változó kisebb, mint a *String* hossza (*len*) akkor visszatér az adott helyen lévő karakter referenciájával, ha a kapott paraméter nagyobb vagy egyenlő a *String* hosszával akkor kivételt dob a függvény.

**inline String operator +(char, const String&):** A függvény balértéke egy karakter, jobbértéke pedig egy *String*. A függvény lefutása után hozzáfűzi a karakterhez a *String*et. Megvalósítás: mivel van karaktert paraméterű konstruktor, így azt felhasználva megvalósítható az, hogy a karakterből *String*et készítünk, utána pedig arra már van operátor, hogy két Stringet összeadjunk, és ezzel az összeg *String*gel visszatérjünk (*inline* függvényként van megvalósítva).

**operator<<(std::ostream&, const String&):** A függvény továbbítja a String karakterláncát a standard outputra, az ostream segítségével.

**operator>>(std::istream&, String&):** A függvény standart inputról érkező adatokat tölti bele az istream segítségével.

**Train_Ticket osztály:**

A Train_Ticket egy absztrakt ősosztály.

**Protected adattagok (elérhetővé kell lenniük a leszármazottak részére):**

**int type:** jegy típusa, azaz első vagy másodosztály

**int train_number :** vonat száma

**int carriage_number :** kocsi száma

**int place :** helyjegy / ülőhely

**String start_place :** járat indulásának helyszíne

**int start_time :** járat indulásának ideje

**String arrival_place:** járat érkezésének helyszíne

**int arrival_time:** járat érkezésének ideje

**Publikus függvények:**

**Train_Tickets() :** A paraméter nélküli konstruktór beállítja az objektumot alapállapotba, ezek az adatok alapvetően érvénytelenek (pl.: -1-re állítja az egész típusú privát adattagokat).

**Train_Tickets(const Train_Tickets &() :**

**save(std::ostream&) :** A függvény egy virtuális függvény, azaz a leszármazottjai még pontosítanak rajta, a Train_Tickets osztálynál továbbítja a birtokában lévő adattagokat az **std::ostream**-re.

**print() :** A függvény egy virtuális függvény, azaz a leszármazottjai még pontosítanak rajta, a Train_Tickets osztálynál kiirja a birtokában lévő adattagokat az **std::cout**-ra.

**operator = (int) :** A függvény balértéke egy Train_Tickets objektum, a jobbértéke pedig egy konstans, a függvényt a leszármazottak pontosítják.

**Train_Tickets & operator + = (int):** A függvény balértéke egy Train_Tickets objektum, a jobbértéke pedig egy konstans, a függvényt a leszármazottak pontosítják.

**Train_Tickets & operator - = (int):** A függvény balértéke egy Train_Tickets objektum, a jobbértéke pedig egy konstans, a függvényt a leszármazottak pontosítják.

**Továbbá minden privát adattaghoz tartozik egy getter, ami visszatér az adott privát adattaggal a függvény meghívása esetén.**

**First_Class osztály:**

A First_Class osztály a Train_Ticket egy leszármazottja.

**Privát adattagok:**

**int dining_table:** megadja a jegyhez tartozó étkezőasztal számát

**int sleeper:** megadja a jegyhez tartozó hálókocsi számát

**Publikus függvények:**

**First_Class():** A paraméter nélkül hívható konstruktor feltölti az objektumot alapvetően érvénytelen adatokkal (pl: egész privát adattagoknak -1 értéket ad)

**First_Class(int, int, int, String, int, String, int, int, int) :** Beállítja a privát adattagokat a paraméterként kapott változókkal.

**operator = (int) :** Egyenlővé teszi az indulási időt és az érkezési időt a balértékként szerepelő konstans változóval, majd visszatér egy First_Class referenciával.

**operator + = (int):** Megnöveli az indulási időt és az érkezési időt a balértékként szerepelő konstans változóval, majd visszatér egy First_Class referenciával.

**operator - = (int):** Csökkenti az indulási időt és az érkezési időt a balértékként szerepelő konstans változóval, majd visszatér egy First_Class referenciával.

**save(std::ostream&) :** Meghívja az ősosztály save függvényét () majd folytatja a leszármazott privát adattagjainak továbbításával az **std::ostream** -re.

**print() :** Meghívja az ősosztály print függvényét () majd folytatja a leszármazott privát adattagjainak kiírásával az **std::cout** -ra.

**isD_t() :** Megadja, hogy az adott jegyhez van-e asztal foglalva.

**isSper() :** Megadja, hogy az adott jegyhez van-e hálófülke foglalva.

**sleeper_fee() :** Visszatér egy integer típusú változóval ami megadja a hálófülke díját. Ha az érkezés korábban van az indulásnál hibát dob.

**Továbbá minden privát adattaghoz tartozik egy getter, ami visszatér az adott privát adattaggal a függvény meghívása esetén.**

**Second_Class osztály:**

A Second_Class osztály a Train_Ticket egy leszármazottja.

**Privát adattagok:**

**bool AC:** megadja, hogy van-e klíma

**bool sleeper:** megadja, hogy van-e büfé kocsi

**Publikus függvények:**

**Second_Class():** A paraméter nélkül hívható konstruktor feltölti az objektumot alapvetően érvénytelen adatokkal (pl: egész privát adattagoknak -1 értéket ad)

**Second _Class(int, int, int, String, int, String, int, bool, bool) :** Beállítja a privát adattagokat a paraméterként kapott változókkal.

**operator = (int) :** Egyenlővé teszi az indulási időt és az érkezési időt a balértékként szerepelő konstans változóval, majd visszatér egy First_Class referenciával.

**operator + = (int):** Megnöveli az indulási időt és az érkezési időt a balértékként szerepelő konstans változóval, majd visszatér egy First_Class referenciával.

**operator - = (int):** Csökkenti az indulási időt és az érkezési időt a balértékként szerepelő konstans változóval, majd visszatér egy First_Class referenciával.

**save(std::ostream&) :** Meghívja az ősosztály save függvényét () majd folytatja a leszármazott privát adattagjainak továbbításával az **std::ostream** -re.

**print() :** Meghívja az ősosztály print függvényét () majd folytatja a leszármazott privát adattagjainak kiírásával az **std::cout** -ra.

**AC_fee():** Visszatér egy integer típusú változóval ami megadja a klíma díját. Ha az érkezés korábban van az indulásnál hibát dob.

**Továbbá minden privát adattaghoz tartozik egy getter, ami visszatér az adott privát adattaggal a függvény meghívása esetén.**

**Vektor osztály:**

A vektor egy generikus osztály, amit a Train_Ticket* változóra tovább lett specializálva.

**Privát adattagok:**

**T * adat :** T típusmutató amely az adatokat tárolja

**size_t meret :** megadja az adatban lévő elemek számát

**Vektor(const T &) :** másoló konstruktor letíltása

**operator = (const T &):** értékadó operátor letíltása

**Publikus függvények:**

**Vektor(size_t):** A konstruktor létrehoz egy vektort, ha nincs paraméter megadva akkor alapértelmezetten a *meret* értéke 0, egyébként lefoglal egy *T* típusú *meret* nagysagú memóriaterületet az *adat*nak.

**T & operator [ ] ( size_t)** : Az indexelő operátor visszaadja egy adott indexen lévő elemet, akkor ha az index létezik. Ellenkező esetben hibát dob.

**T const& operator [ ] (size_t) const:** Konstansokra működő indexelő operátor.

**push_back (T):** Új elemet ad hozzá az objektumhoz, létrehoz egy átmeneti változót, aminek mérete eggyel nagyobb mint az eredeti tárolónak, majd az

átmenetibe belemásolja az eredetiben tárolt adatokat, majd az átmeneti tároló végéhez adja az újonnan érkező elemet. Ezt követően törli az eredeti adatokat. Utolsó lépésben pedig átadja az átmeneti tároló adatait az eredetinek és megnöveli az eredeti méretet eggyel.

**print()**: A függvény kiirja a tárolóban lévő összes adatot az **std::cout** -ra.

**Vektor<Train_Tickets*> specifikus függvények:**

**clear()**: A heterogén kollekcióhoz foglalt elemek felszabadítását végzi el a függvény

**erase(String, String) :** A függvény a paraméterül kapott indulási és érkezési helyszín alapján törli az adatbázis elemeit. Ha 0 találat van a paraméterek alapján, akkor hibát dob.

**seek_ido(int) :** A megadott idő paramétert (int) felhasználva kilistázza azokat a jegyeket, amelyeknek az indulási vagy érkezési ideje megegyezik a paraméterrel, ha nincs ilyen adat, akkor hibát dob.

**seek_hely(String) :** A megadott hely paramétert (String) felhasználva kilistázza azokat a jegyeket, amelyeknek az indulási vagy érkezési ideje megegyezik a paraméterrel, ha nincs ilyen adat, akkor hibát dob.

**valtozas(String, String, int, int, int, bool) :** A függvény paraméterül kap egy indulási és egy érkezési helyszínt (Stringek) továbbá egy indulási és egy érkezési idő (int). Ezek alapján megkeresik azokat az elemeket, amelyekre illenek a paraméterek, majd a paraméterlistán 5. elemként átadott változás paraméterrel megváltoztatják a menetidőt. Azt, hogy az adott járat késik vagy hamarabb érkezik azt a bool változó adja meg, ha igaz, akkor késik, ha hamis, akkor siet.

**print():** A print függvényt a heterogén kollekcióra specializálni kellett, mivel nem a mutató címeket kell kiiratni, hanem az adott jegyek adatait. Így kiíráskor minden elemre meghívja annak saját print függvényét.

**save(std::ostream&):** A heterogén kollekció fájlba mentésének segítését szolgáló függvény, a heterogén kollekció minden elemére meghívja annak a Train_Tickets osztály és leszármazottainál található a mentés formátumának megfelelő függvényt.

**Kezelofelulet osztály:**

**Privát adattagok:**

**Vektor<Train_Tickets*> w :** tárolja az összes használatban lévő adatot

**Publikus függvények:**

**Kezelofelulet():** Elvégzi az adatok beolvasását a jegyek.txt fájlból, majd az adatokat a *push_back()* függvény segítségével hozzáadja a heterogén kollekcióhoz. Ha a fájl nem nyílik meg vagy üres, akkor a program hibát dob.

**ment():** A vektorban és a Train_Tickets-ben definiált függvények segítségével kimenti az adatokat a jegyek.txt-be.

**torles():** A **Vektor<Train_Tickets*>** -ban definiált törlő függvény segítségével törli azokat az elemeket az adatbázisból, amelyek megegyeznek a kritériumokkal.

**keres_hely():** Kilistázza azokat a jegyeket, amelyek megfelelnek a függvényen belül megadott paraméternek, a keresést a **Vektor<Train_Tickets*>** -ban definiált függvény végzi el.

**keres_ido():** Paramétereket kér be a felhasználótól, amíg azok meg nem felelnek egy valós időnek, majd a **Vektor<Train_Tickets*>** -ban definiált függvény segítségével jelzi a felhasználónak a keresés eredményét.

**adatfelvetel():** A függvény paramétereket kér be a felhasználótól, addig amíg azok meg nem felelnek a kritériumoknak (pl.: helyjegy száma nem lehet negatív) majd a *push_back()* függvény segítségével hozzáadják az adatokat tároló vektorhoz.

**keses(bool pred = true) :** Itt kezeli a program a késéseket, a pred változódönti el, hogy a vonat éppen késik, vagy siet, az adatok tényleges megváltoztatását a **Vektor<Train_Tickets*>** -ban végzi el. A hozzá szükséges adatokat addig olvassa be a program, amíg minden adat a neki megfelelő határok közé nem esik (pl.: a vonat száma nem lehet negatív).

**print():** A **Vektor<Train_Tickets*>** -ban definiált print() függvény segítségével ír az **std::cout** -ra.

**fomenu():** Itt valósul meg az állapotgép, a műveletek kiválasztása

**kezeles(Allapot) :** Az állapotgép aktuális állapotát kezeli

**nextstate(Allapot,int) :** Az állapotgép következő állapotát kezeli

**test():** A feladathoz tartozó teszteseteket tárolja

**percben():** A felhasználótól bekér egy időpontot órában és percben majd azt visszaadja perc formátumban

**Kapcsolódó függvények:**

**atalakit(std::string):** A paraméterül kapott sztringet átalakítja egy egész szám típusú változóvá, majd visszatér vele. A függvény fő célja a főmenüből érkező adatok feldolgozása.

**Egyéb változók:**

**enum Allapot :** az állapotgép állapotait tárolja.

**Fájlformátum:**

A fájlban az adatok egymástól szóközzel vannak elválasztva az alábbi sorrendben:

1. **típus (int)**
2. **vonat (int)**
3. **kocsi (int)**
4. **helyjegy (int)**
5. **indulás helyszíne (String)**
6. **indulás ideje percben (int)**
7. **érkezés helyszíne (String)**
8. **érkezés ideje percben (int)**
9. **étkezőkocsi száma (int) / klíma (bool)**
10. **hálókocsi száma (int) / büfé (bool)**

```
1 15 30 27 Budapest 500 Kecskemet 600 13 15
2 3 55 11 Baja 135 Szeged 333 1 0
```

**Felhasználói dokumentáció**

A vonatjegy program egy vonatjegy adatbázis adatainak kezelésére való. Ezt az adatbázist tudja a felhasználó kezelni.

A program főmenüje egy konzol ablakban jelenik meg. A felhasználó számjegyek megadásával tud tevékenységet választani a főmenüből:


Bizonyos műveletek az adatbázis módosításával járnak ilyen a jegy felvétele vagy a jegy törlése, ekkor a program a művelet elvégzése után automatikusan elmenti a módosított adatbázist a célfájlba.

Az "OSSZES KILISTAZASA" menüpont választásával a program kiírja az adatbázisban található összes adatot.

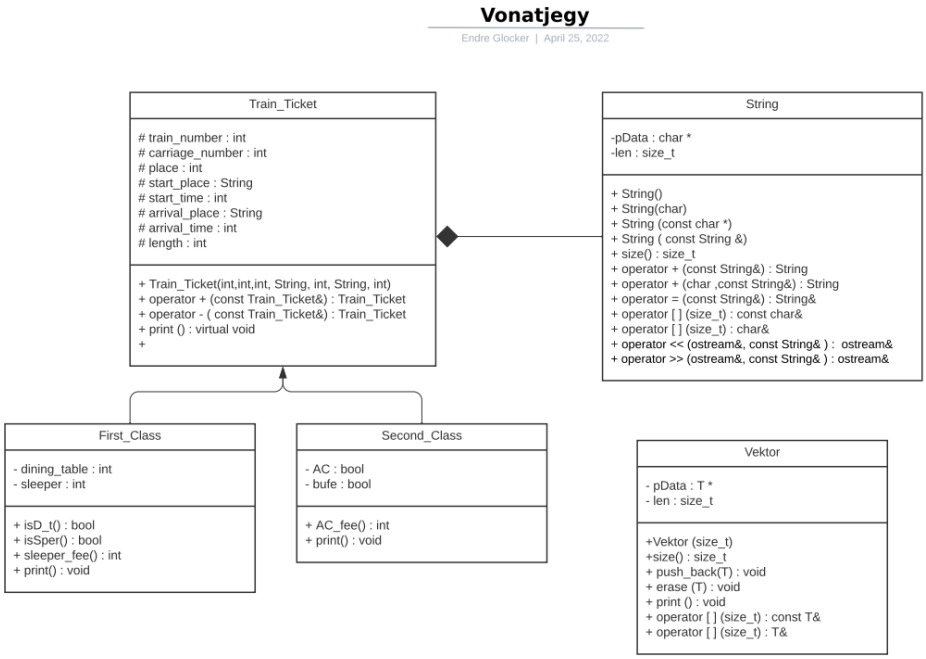
A "JEGY TORLESE " pont alatt a felhasználónak meg kell adnia egy érkezési és egy indulási helyszínt, ha az adatbázisban van ezzel egyező elem akkor azt a program törli, majd elmenti a módosított adathalmazt, ha nem sikerült egyezést találni az adatok között, akkor a program azt egy hibaüzenettel közli a felhasználóval.

A "JEGY FELVETELE" menüpontban a felhasználónak meg kell adnia egy új vonatjegy összes adatát, egy adatról addig nem tud továbblépni a felhasználó, amíg az meg nem felel a kritériumoknak például: az idő nem lehet negatív. Ha minden adatot jól megadta, akkor a program felveszi az új jegyet az adatbázisba, majd elmenti azt.

"KERESES (JEGY) " menüpontnak két almenüje található a " KERESES HELY ALAPJAN" és a "KESRESES IDO ALAPJAN ", előbbinél egy helyszínt kell megadni utóbbinál egy időpontot órában és percben (ha megfelelő időformátumot ad meg), ha azok megtalálhatók az adatbázisban akkor a program kilistázza azokat, ha nem, akkor hibaüzenetet ír a konzolba.

" KESESEK KEZELESE" szintén két almenüponttal rendelkezik a " KESES MEGADASA" és a "KORABBAN ERKEZES BEALLITASA" mindkét esetben a felhasználónak meg kell adnia a járat adatait (indulás, érkezés helyszíne és ideje) valamint egy perc formátumú időt, amivel az első menüpontban a járat idejét megnöveli, a másodikban pedig a járat idejét csökkenti vele.

**Vonatjegy pontosított specifikáció**




## Főmenü:

A főmenüt egy véges állapotgép alkotja, az állapotokat egy enum-ban tároljuk, és az állapotok közötti váltakozást egy switch-case biztosítja.

## Beolvasás (fájlból):

Ha már létezik kezdőadat egy text formátumú fájlból történik a beolvasás, az algoritmus soronként olvassa a fájlt, majd az onnan származó adatokat egy Train_Ticket Vektorban tárolja el, alapértelmezetten az adatok szóközzel vannak elválasztva egymástól. Az adatok a Train_Ticket osztály privát adattagjaihoz hasonló sorrendben vannak szétválogatva, és aszerint vannak hozzáadva a Vektorhoz.

## Kiírás (fájlba):

A fájlba történő kiírás során egy text állományt hozunk létre, amelybe a Vektorban található összes objektum adatát soronként kiírjuk, az adattagokat egymástól szóközzel elválasztva, egy objektum adatai egy sort képeznek, végét enterrel lezárva.

## String osztály:

**String() paraméter nélküli konstruktor** : létrehoz egy NULL pointer char * -ot, aminek hossza 0.

**String(char)** : létrehoz egy 1 karakter hosszú tömböt.

**String(const char *)** : létrehoz egy karakter tömböt, aminek elemei const char * , és hossza megegyezik vele.

**String(const String&)** : másoló konstrukto

**size()** : visszaadja az adott String hosszát

**operator +** : összefűz egy karaktert egy String-gel vagy String-et String-gel pl: "a" + "lma" = "alma" vagy "almas" + "pite" = "almaspite"

**operator []** : visszaadja egy adott indexű karaktert a String-ben, ha az létezik, ha nem, akkor kivételt ad

**operator <<** : kiírja a String-et az ostream-re

**operator >>** : beolvas egy String-et az ostream-ről

## Vektor osztály:

A vektor osztály egy generikus adatszerkezet, amely az adatok tárolását szolgálja.

**Vektor()** : létrehoz egy üres Vektort

**size()** : visszaadja a jelenlegi méretet

**push_back()** : növeli a Vektor elemszámát 1-gyel, majd T-t belemásolja az új helyre

**erase()** : megkeresi azokat a T-ket, amire a keresett tulajdonság vonatkozik, majd törli azokat a vektorból.

**print()** : kiírja a vektor elemeit

**operator [ ]** : visszaadja egy adott indexű elemet

## Train_Ticket osztály:

**Train_Ticket()** : inicializálják az elemeket (ha szükséges akkor alapértéket adnak neki)

**get_smth()** : visszaadja az adott adattag értékét

**set_smth()** : beállítja az adott adattag értékét

**print()** : kiírja a Train_Ticket összes adattagját

**operator +** : ha a vonat késik, akkor a késés idejét ezzel lehet beállítani

**operator -** : ha a vonat hamarabb érkezne, annak érkezési idejét lehet beállítani

## First_Class osztály:

**isD_t()** : meghatározza, hogy az utas rendelkezik-e ebédlőasztal foglalással.

**isSper()** : meghatározza, hogy az utas az adott út során rendelkezik-e hálókocsi fülkével

**sleeper_fee()** : megadja a hálófülkéért fizetendő díjat

## Second_Class osztály:

**AC_fee()** : kilométer arányosan meghatározza az AC árát.

**Feladat és tervezet**

## 1.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!

## 2. SPECIFIKÁCIÓ:

A feladat egy vonatjegy eladó rendszer tervezése, objektum orientált szemléletmóddal megvalósítva. A feladat során keletkező adatokat file-okba lesznek elmentve, és onnan is lesznek beolvasva, ha már keletkezett korábbi mentés vagy léteznek kezdőadatok. A jegyeket kezelő osztály képes lesz bizonyos operátorok átdefiniálására specifikus adatok esetén. (pl.: operator+(int kesedelem) vagy operator-(int elobb) ) Valamint a program fő kezelőfelületét egy konzolablak fogja nyújtani, ahonnan kiválaszthatók a program menüpontjai. A program adatszerkezete kizárólag az angol abc karaktereit tudja helyesen kezelni. Csak olyan adatokkal lehet az elkészített tömböt használni, melyre értelmezve van az értékadás művelete. A hibás adatokat hibakezelése kivétel átadásával (const char*) valósulnak meg (pl.: indulási idő: -1:23)