HÁZI FELADAT

Programozás alapjai 2.

Végleges

Berényi Henrik Dániel
QP4TVJ

2021. április 24.

Tartalom

Tartalom

1. Feladat 2
2. Feladatspecifikáció 2
3. Pontosított feladatspecifikáció 3
4. Terv 4
   4.1. Objektum terv 4
   4.2. Algoritmusok 5
      4.2.1. Jaratok osztály beolvas függvénye a fájlból való beolvasáshoz 5
      4.2.2. Add függvény a heterogén kollekcióhoz való hozzáadáshoz. 6
      4.2.3. Remove függvény a heterogán kollekcióból való eltávolításhoz. 6
      4.2.4. Tesztprogram algoritmusai. 7
5. Megvalósítás 8
   5.1. Jegy osztály és annak tárolója, a Jegyek osztály bemutatása 9
   5.2. Vonat osztály és annak tárolója, a Jaratok osztály bemutatása 10
   5.3. Datum osztály bemutatása 11
6. Tesztelés 12
   6.1. Memóriakezelés tesztje 12
   6.2. Lefedettségi teszt 12

1. 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!

2. Feladatspecifikáció

A feladat egy vonatjegy eladó rendszer megvalósítása.

A feladat előírásának megfelelően megvalósítom a vonatok felvételét és a jegy kiadását. A feladat szövege nem írja, hogy hány jegyet, illetve vonatot kell tárolnia a programnak, így én amellett döntöttem, hogy dinamikus memóriakezeléssel tetszőleges számú vonatot lehet felvenni, illetve jegyet eladni.

Előírás, hogy a jegyekkel végezhető műveletek overloadolva legyenek, ezek közül a másolást, értékadást és kiírást valósítom meg.

A program menüvezérelt, a felhasználó a menüpontok számaival navigálhat a különböző menüpontok között.

A program indulásakor képes lesz egy fájlban előre eltárolt vonatok adatainak beolvasására, majd később a standard inputról való hozzáadásra is.
A program a vonatok és a jegyek hozzáadásakor az inputokat ellenőrzi és hibás adat megadása esetén(pl.: nem létező dátum, rossz járatszám formátum) kivételt dob.
A futást követően az új adatok a bemenetként használt fájlba mentődnek el.

A programhoz teszteseteket is készítek majd, amik helyes működés mellett a hibakezelést is ellenőrizni fogják.

3. Pontosított feladatspecifikáció

A feladat egy vonatjegy eladó rendszer megvalósítása.

A program képes vonatok és megvásárolt jegyek adatainak tárolására dinamikusan allokált memóriaterületen.

A vonatok és a jegyek adatai 1-1 .txt fájlban vannak eltárolva, amelyekből a program az indulásakor kiolvassa az összes információt és a futását követően eltárolja a változásokat is.

A szöveges fájlok adatainak formátuma a következő:
vonatok.txt:
azonosító#év#hónap#nap#óra#perc#év#hónap#nap#óra#perc#Indulási_állomás#Érkezési_állomás
jegyek.txt: név#kocsiszám#ülésszám#azonosító

Az adatok formátumai:
azonosító: a vonat azonosítója, 3 nagy betű és 3 szám ebben a sorrendben
év/hónap/nap/óra/perc: indulási és érkezési dátum azonosításához szükséges adatok, egész számok
indulási állomás/érkezési állomás/név: tetszőleges hosszú string (lehet benne szóköz)
kocsiszám/ülésszám: vonat kocsiját és az ülést azonosító számok

A program 6 osztállyal rendelkezik: a vonat osztály a vonatok adatait tárolja, a jegy osztály pedig a jegyek adatai mellett egy vonat objektumot is tartalmaz. Létezik egy Datum osztály is, amely tárolhatja egy járat indulási vagy érkezési időpontját. A szöveges adatokat a String osztály tárolja. Az utolsó két osztály heterogén tárolók, amelyek a Vonat és a Jegy objektumokat tárolják dinamikusan.

A Jegy osztályban az implicit függvények és a konstruktorok mellett a következő operátorok vannak definiálva:
- Értékadás(másoló konstruktorral együtt)
- Kiíró operátor(<<)

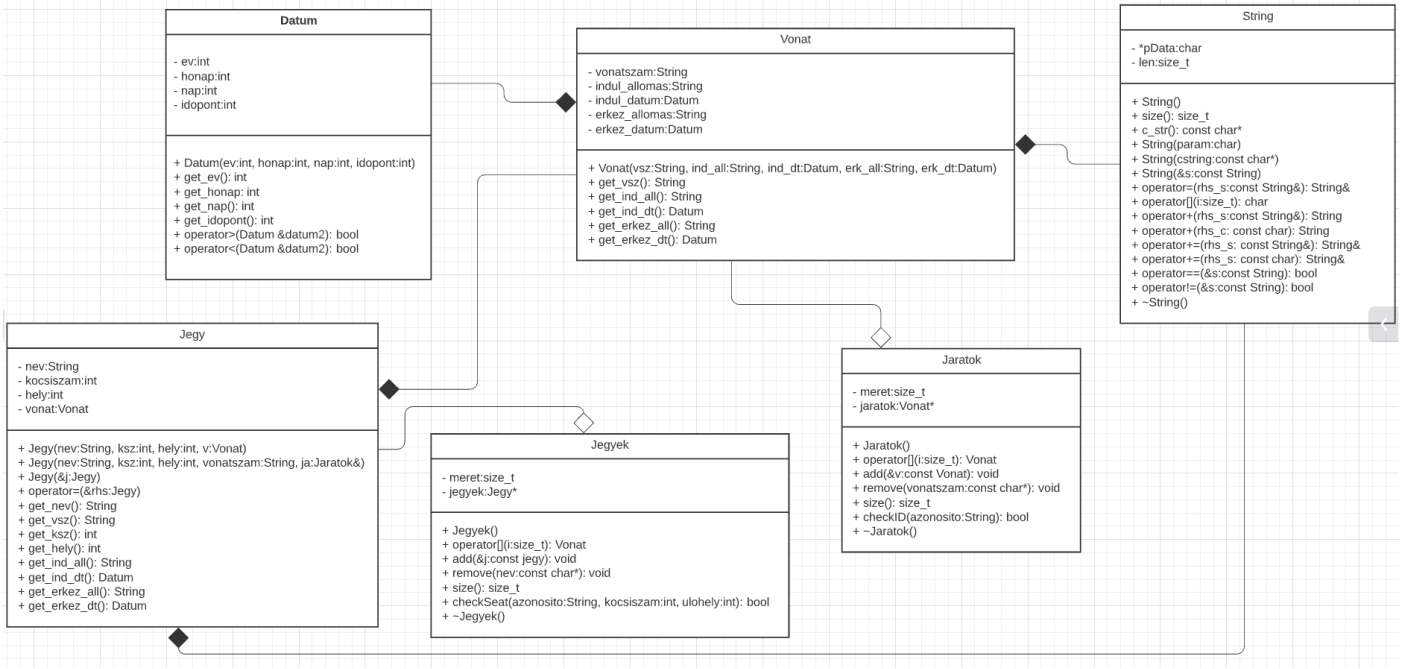
A program képes ellenőrizni a megadott inputok helyességét az azonosító, kocsi- és ülésszám és a dátumok esetében.
A program a fájlokból beolvasott adatokat nem ellenőrzi, azokat minden esetben helyesnek feltételezi, hiszen azt csak a program módosíthatja és íráskor csak a helyes adatokkal rendelkező objektumokat rögzíti.

4. Terv

A feladat az objektumok és a tesztprogram megtervezését igényli.

4.1. Objektum terv

Az objektumok részletes leírása a 3. fejezetben lévő pontosított specifikációban olvasható.



4.2. Algoritmusok

4.2.1. Jaratok osztály beolvas függvénye a fájlból való beolvasáshoz

```
void beolvas(Jaratok& j, const char* fajlnev) {
    ifstream fpp;
    fpp.open(fajlnev);
    if(!fpp) {
        printf( _Format: "KIVÉTEL\n");
        throw "QP4TVJ";
    }
    String* adatok = new String[13];

    int i = 0;
    while(getline( &: fpp, &: adatok[i], d: '#')) {
        if(i == 11) {
            getline( &: fpp,  &: adatok[12],  d: '\n');
            Datum ind( ev: stoi( &: adatok[1]),  honap: stoi( &: adatok[2]),  nap: stoi( &: adatok[3]),
                       idopont: stoi( &: adatok[4])*60+stoi( &: adatok[5]));

            Datum erk( ev: stoi( &: adatok[6]),  honap: stoi( &: adatok[7]),  nap: stoi( &: adatok[8]),
                       idopont: stoi( &: adatok[9])*60+stoi( &: adatok[10]));

            Vonat v( vsz: adatok[0],  ind_all: adatok[11], ind,  erk_all: adatok[12], erk);
            i = -1;
            j.add(v);
            delete[] adatok;
            adatok = new String[14];
        }

        i++;
    }
    delete[] adatok;
    fpp.close();
}
```

Ez a függvény beolvassa az eltárolt járatok adatait a megadott nevű fájlból, felhasználva a String osztályra definiált getline() és stoi() függvényeket. A getline() és stoi() függvények pontosan úgy működnek, mint az std::string hasonló nevű függvényei.

Beolvas függvény a Jegyek osztály esetén is létezik, azonban az a beolvasott adatokat kivéve nem különbözik az előbbitől.

4.2.2. Add függvény a heterogén kollekcióhoz való hozzáadáshoz.

```
void Jaratok::add(const Vonat& v) {
    Vonat* temp = new Vonat[meret+1];
    for(size_t i = 0; i < meret; i++) {
        temp[i] = jaratok[i];
    }
    temp[meret] = v;
    delete[] jaratok;
    jaratok = temp;
    meret++;
}
```

Add függvény a Jegyek osztály esetén is létezik, azonban az a beolvasott adatokat kivéve nem különbözik az előbbitől.

4.2.3. Remove függvény a heterogán kollekcióból való eltávolításhoz.

```
void Jaratok::remove(const char* vonatszam) {
    for(size_t i = 0; i < meret; i++) {
        if(jaratok[i].get_vsz() == vonatszam) {
            Vonat* temp = new Vonat[meret-1];
            size_t idx = 0;
            for(size_t j = 0; j < meret; j++) {
                if(jaratok[j].get_vsz() != vonatszam) {
                    temp[idx] = jaratok[j];
                    idx++;
                }
            }
            meret--;
            delete[] jaratok;
            jaratok = temp;
            return;
        }
    }
    throw "Nincs ilyen vonatszam!\n";
}
```

Remove függvény a Jegyek osztály esetén is létezik, azonban az a beolvasott adatokat kivéve nem különbözik az előbbitől.

4.2.4. Tesztprogram algoritmusai.

A programhoz készül majd egy tesztprogram is, amely a program összes függvényének működését teszteli a menü kivételével. A program ugyanis menüvezérelt és inputokat vár el a felhasználótól, majd a különböző menüpontok hívják meg a megfelelő függvényeket. A tesztprogrammal azonban sokkal előnyösebb egyenként tesztelni a függvényeket, megvizsgálni azok hibakezelését és működését.

5. Megvalósítás

A feladat megoldása a tervezésben előírt osztályokat és tárolókat igényelte. Az osztályok végleges interfésze a tervezési lépésben meghatározott módon alakult. A tervezéshez képest annyi változott, hogy a dátum nagyobb/kisebb, mint operátorait lecseréltem nagyobb/kisebb egyenlő operátorokra, hiszen egy vonat keresésénél akkor van találat, ha éppen ugyanazt a dátumot adjuk, meg, mint amikor a keresett vonat indul/érkezik.
A program több külön állományra van bontva osztályok és más funkciók szerint. A tesztesetek az NHF3-hoz képest kibővültek, a Jporta-ra feltöltött szabványos bemenettel együtt pedig a program összes létező függvényét tesztelik.
A továbbiakban a bemutatom a fontosabb interfészeket és algoritmusokat a program forrása alapján generált dokumentáció felhasználásával.

5.1. Jegy osztály és annak tárolója, a Jegyek osztály bemutatása

A Jegy osztály a program futása során megvásárolt vonatjegyek tárolására alkalmas. Mivel minden vonatjegy egyben helyjegy is, ezért az ülőhelyet is eltárolja.
Az osztály rendelkezik paraméter nélküli és másoló konstruktorral, valamint példányosítható egy Vonat objektum segítségével, illetve egy vonatazonosító és egy Jaratok heterogén kollekció alapján is.
Az osztály az értékadás és az egyenlőség összehasonlító operátort használja overloadolva.

Publikus tagfüggvények

	Jegy (**String** nev, int ksz, int hely, **Vonat** v)
	Jegy (**String** nev, int ksz, int hely, **String** vonatszam, **Jaratok** &ja)
	Jegy (const **Jegy** &j)
	Jegy ()
Jegy &	**operator=** (const **Jegy** j)
bool	**operator==** (const **Jegy** &j) const
String	**get_nev** () const
String	**get_vsz** () const
int	**get_ksz** () const
int	**get_hely** () const
String	**get_ind_all** () const
Datum	**get_ind_dt** () const
String	**get_erkez_all** () const
Datum	**get_erkez_dt** () const
Vonat	**get_vonat** () const

A Jegyek egy heterogén kollekció, amely Jegy objektumok tárolására alkalmas. Csak paraméter nélküli konstruktora van. Objektumokat hozzáadni az *add* függvénnyel, objektumokat eltávolítani pedig a *remove* függvénnyel lehet. Az osztály ezeken a függvényeken kívül rendelkezik még egy, a méretet visszaadó *size* függvénnyel és egy ülés foglaltságát ellenőrző *checkSeat* függvénnyel is. Az osztály az indexelő operátort használja overloadolva.

Publikus tagfüggvények

	Jegyek ()
Jegy	**operator[]** (size_t i) const **Jegyek**. Részletek...
void	**add** (const **Jegy** &j)
void	**remove** (const char *nev)
bool	**checkSeat** (**String** azonosito, int kocsiszam, int ulohely) const
size_t	**size** () const
	~Jegyek ()

5.2. Vonat osztály és annak tárolója, a Jaratok osztály bemutatása

A Vonat osztály a program futása során használt vonatok adatainak tárolására alkalmas.
Az osztály rendelkezik paraméter nélküli konstruktorral, valamint példányosítható a vonat adataival.
Az osztály az egyenlőség összehasonlító operátort használja overloadolva.

Publikus tagfüggvények

	Vonat (**String** vsz, **String** ind_all, **Datum** ind_dt, **String** erk_all, **Datum** erk_dt)
	Vonat ()
bool	**operator==** (const **Vonat** &v) const
String	**get_vsz** () const
String	**get_ind_all** () const
Datum	**get_ind_dt** () const
String	**get_erkez_all** () const
Datum	**get_erkez_dt** () const

A Jaratok egy heterogén kollekció, amely vonat objektumok tárolására alkalmas. Csak paraméter nélküli konstruktorral rendelkezik, elemeket hozzáadni, illetve elvenni az *add* és *remove* függvényekkel lehetséges. Az osztály ezeken a függvényeken kívül rendelkezik még egy, a méretet visszaadó *size* függvénnyel és járatazonosítót ellenőrző *checkID* függvénnyel is. Az osztály az indexelő operátort használja overloadolva.

Publikus tagfüggvények

	Jaratok ()
Vonat	**operator[]** (size_t i) const Járatok. Részletek...
void	**add** (const **Vonat** &v)
void	**remove** (const char *vonatszam)
size_t	**size** () const
bool	**checkID** (**String** azonosito) const
	~Jaratok ()

5.3. Datum osztály bemutatása

A Datum osztály feladata egy adott járat indulási vagy érkezési időpontjának tárolása. Az osztály tartalmazza az évet, a hónapot, a napot, az órát és a percet is. Az óra és perc a könnyebb tárolás miatt összevonva, *idopont* néven tárolódik és csak kiíráskor lesz szétbontva órára és percre. Az osztály rendelkezik paraméterek nélküli konstruktorral, valamint példányosítható a fentebb említett értékekkel. Az osztály a nagyobb egyenlő, a kisebb egyenlő és az egyenlőség összehasonlító operátorokat overloadolja.

Publikus tagfüggvények

	Datum (int ev, int honap, int nap, int idopont)
	Datum ()
int	**get_ev** () const
int	**get_honap** () const
int	**get_nap** () const
int	**get_idopont** () const
bool	**operator>=** (**Datum** &datum2) const
bool	**operator<=** (**Datum** &datum2) const
bool	**operator==** (const **Datum** &datum2) const

6. Tesztelés

A tesztelés kétféleképpen történik: a Jporta először lefuttatja a program menüvezérelt részét a megadott szabványos bemeneti adatokkal az összes lehetséges menüponton végighaladva. A menü tesztelése után a Jporta kilép a menüből és lefutnak a menü utáni tesztesetek. A menü utáni tesztesetek gtest_lite segítségével vannak megírva. Feladatuk, hogy külön-külön teszteljék a program függvényeit és overloadolt operátorait.

6.1. Memóriakezelés tesztje

A memóriakezelés ellenőrzését a laborgyakorlatokon használt MEMTRACE modullal végeztem. Ehhez projekt szinten definiáltam a MEMTRACE compiler definitiont.
Memóriakezelési hibát nem tapasztaltam a futtatások során, valamint az előzetes Jporta teszt feltöltésén is *Sikeres memóriaszivárgás teszt!* kimenetet kaptam.

6.2. Lefedettségi teszt

A szabványos kimeneti és a funkcionális tesztek a program minden ágát lefedték.
A Jporta *100% overall code coverage* kimenetet produkált.
Egyedül a menu.hpp esetében írt 0%-ot, de abban a header fájlban csak egy függvénydefiníció van, ami azonban használva van a program futása során.