NHF3 – Skeleton

Hubai Rajmund Szilveszter CQFKI2

Egy nyilvántartásjellegű programot, azon belül a „Vonatjegy” eladó rendszer megvalósítását választottam. A program képes lesz vonat összes CRUD funkcióját ellátni (Create, Read, Update, Delete), menetrend elkészítésére, jegy kiadására.

A program indításakor egy menü fogad minket, kilistázza az egyes menüpontokat, és azokat a karaktereket, amelyekkel tudja a felhasználó kezelni a programot. Tehát ha mondjuk lenyomjuk az „1”-es billentyűt, akkor megjeleníti az „Új jegy felvétele” menüpontot. Az alábbi módon fog kinézni a felület:

```
(1.) Új jegy felvétele
(2.) Menetrendek
(3.) Vonat testreszabása
(4.) Kilépés

A menüpont kiválasztásához, nyomja meg a hozzátársított számot!
```

Amint a felhasználó megnyomta az adott számot, amit szeretett volna, egy új menüpont fog megjelenni az alábbi módon. Jelen esetben az „Új jegy felvétele” menüpont után van a program:

```
Kérjük az utas nevét!
Kérjük a kezdő állomás nevét!
Kérjük a végső állomás nevét!
```

Ekkor a program sorra lekérdezi a felhasználótól a program, hogy az adott paraméterbe, milyen adatokat szeretne beírni. Természetesen, ahol számot kér, és a felhasználó szöveget ír be, azt kezeli a program úgy, hogy még egyszer felszólítja a felhasználót, hogy helyesen írja be a paramétert.

Amint a felhasználó megnyomta az adott számot, amit szeretett volna, egy új menüpont fog megjelenni az alábbi látható módon. Jelen esetben a „Menetrendek” menüpont után van a program:

```
(1.) Új menetrend hozzáadása
(2.) Meglévő menetrend szerkesztése
(3.) Menetrendről bővebben
(4.) Vonat törlése

A menüpont kiválasztásához, nyomja meg a hozzátársított számot!
```

Új menetrend hozzáadása

Ekkor a felhasználótól sorra bekéri a paramétereket a program az alább látható módon. A felhasználónak ezeket egyesével be kell írnia, végül amint megvan minden adattal, amit kér a program, elmenti azt, és később meg is tudja nyitni.

```
Kérjük az állomás nevét!
Kérjük a vonat nevét!
Kérjük a megállóba érkezést!
```

Meglévő menetrend szerkesztése

Ekkor a felhasználótól a program lekérdez egy allomasID-t, természetesen ha nincs az a meglévő menetrend, akkor hibát ad. Ha sikeresen beírt egy allomasID-t, akkor megnyílik a következő menüpont, majd kilistázza, mely paramétert szeretné szerkeszteni az alább látható módon

```
Kérem az állomás egyedi azonosítóját! GYTP
Melyiket szeretnéd módosítani?
(1.) Állomás egyedi azonosítója
(2.) Vonat neve
(3.) Megállóba érkezés

A menüpont kiválasztásához, nyomja meg a hozzátársított számot!
```

Menetrendről bővebben

Ekkor a felhasználótól a program lekérdez egy allomasID-t, természetesen ha nincs az a meglévő menetrend, akkor hibát ad. Ha sikeresen beírt egy allomasID-t, akkor megnyílik a következő menüpont, ekkor a program tagolva, tördelten kiírva a menetrend paramétereit az alább látható módon:

```
Kérem az állomás egyedi azonosítóját! GYVA
    - Állomás neve: Győr
    - IC994:
        o Hétköznap:
            ▪ Hétfő: minden óra 20 perckor indul Győr felé
            ▪ Hétfő: minden óra 35 perckor indul Tata felé
            ▪ …..
    - S10:
        o ….
```

Amint a felhasználó megnyomta az adott számot, amit szeretett volna, egy új menüpont fog megjelenni az alábbi látható módon. Jelen esetben a „Vonat testreszabása” menüpont után van a program:

(1.) Új vonat hozzáadása
(2.) Meglévő vonat szerkesztése
(3.) Vonatról bővebben
(4.) Vonat törlése

A menüpont kiválasztásához, nyomja meg a hozzátársított számot!

Új vonat hozzáadása

Ekkor a felhasználótól sorra bekéri a paramétereket a program, majd eltárolja az alább látható módon:

Kérjük a vonat egyedi azonosítóját!
Kérjük a vonat nevét!
Kérjük a megállókat!
…

Meglévő vonat szerkesztése

Ekkor a felhasználótól a program lekérdez egy vonatID-t, természetesen ha nincs az a meglévő vonat, akkor hibát ad. Ha sikeresen beírt egy vonatID-t, akkor megnyílik a következő menüpont, majd kilistázza, mely paramétert szeretné szerkeszteni az alább látható módon:

Kérem a vonat egyedi azonosítóját! IC994
Melyiket szeretnéd módosítani?
(1.) Vonat egyedi azonosítója
(2.) Vonat típusa
(3.) …
A menüpont kiválasztásához, nyomja meg a hozzátársított számot!

Vonatról bővebben

Ekkor a felhasználótól a program lekérdez egy vonatID-t, természetesen ha nincs az a meglévő vonat, akkor hibát ad. Ha sikeresen beírt egy vonatID-t, akkor megnyílik a következő menüpont, ekkor a program tagolva, tördelten kiírva a vonat paramétereit az alább látható módon:

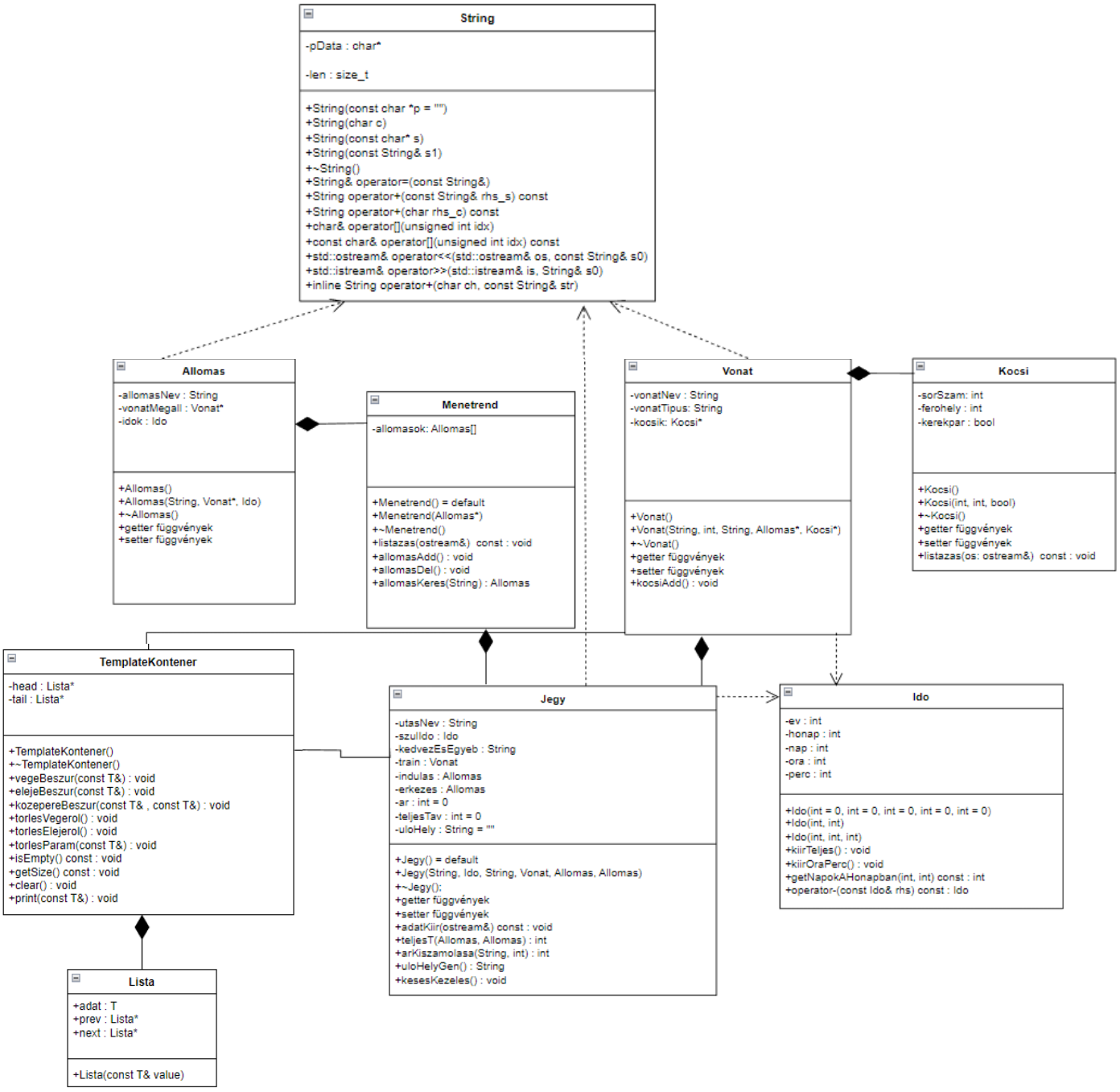
Kérem a vonat egyedi azonosítóját! IC994

-Típus:	Intercity
-Vonat neve:	Savaria Intercity 994
-Vonat kezdőpontja:	Budapest-Keleti
-Vonat végállomása:	Szombathely

…

Az osztályok közötti kapcsolat

A program az alábbi módon fog működni, egyelőre osztályok közötti hivatkozások



bemutatását az alábbi UML diagram mutatja:

A program „.txt” kiterjesztésű fájlokban fogja tárolni az adatokat, ezeket elindulásakor a program beolvassa, majd láncolt listában tárolja, hogy ne kelljen újra kiolvasni a fájlból az adatokat, amikor csak használva van. A fájlokból „**;**”-tal elválasztva, az alábbi látható módon a jegyek lesznek eltárolva:

utasNev;szulIdo;kedvez;train;indulas;erkezes;ar;teljesTav;uloHely

A vonatjegyek a való élettel ellentétben kizárólag csak egyetlen vonatra érvényesek, mivel minden esetben fel kell tüntetni a helyjegyet. A „kedvez” paraméterrel a program kezelni fogja az egyes kedvezményeket.

Paraméterek – zárójelben jelölve, milyen változóval lesz eltárolva a programban

- `utasNev`: utas neve, pl.: Gipsz Jakab (String)
- `szulIdo:` az utas születési ideje (Ido)
- `kedvezEsEgyeb:` kedvezmények (lásd később) (String)
- `train:` vonat neve (String)
- `indulas`: az indulási állomás, ahonnan szól a jegy (String)
- `erkezes:` az érkezési állomás, ameddig szól a jegy (String)
- `ar:` a jegy ára Ft-ban értve, pl.: 1250 (int)
- `teljesTav:` út alatt megtett teljes távolság (lásd később) (int)
- `uloHely:` utas ülőhelye (lásd később) (String)

A kedvezményeket úgy oldom meg, hogy van egy tömbünk, mindegyik indexen egy különböző kedvezmény van, ha kiválaszt a felhasználó egy kedvezményt, akkor a „kedvez” attribútumba ír egy „1”-est, különben „0”. Ezeket egymáshoz fűzi, például: Diák, Nyugdíjas, BKK bérlet, MÁV bérlet. Ha a felhasználó diák, nem nyugdíjas, nincs BKK bérlete, viszont van MÁV bérlete, akkor a „kedvez” attribútum „1001” lesz.

A teljes távot úgy oldom meg, hogy mivel az állomások láncolt listában tárolom el az állomásokat (lásd később), ezért a számolás transzparenciája érdekében – az ár figyelembe veszi a távolságot –, ezért minden állomás között 10 km van. Tehát ha az állomások úgy vannak eltárolva, hogy „Győr”, „Budapest”, „Szolnok”, és a felhasználó Győr->Szolnok távolságot szeretné megtenni, akkor ez 20 km-nek felel meg, mivel Győrből Budapest 10 km, Budapesttől Szolnok 10 km.

Az ülőhely formátumát úgy oldom meg, hogy három számjegyű, Stringként eltárolt szám lesz, az első számjegy a kocsi száma, a maradék két fennálló számjegy a kocsiban lévő hely.

A fenti paraméterek egy jegyhez szólnak, természetesen a vonatokhoz is ehhez hasonló módon lesz eltárolva, szintúgy az alább látható módon:

vonatNev;vonatTipus;allomasok;kocsik

Paraméterek – zárójelben jelölve, milyen változóval lesz eltárolva a programban

- `vonatNev:` az adott vonat neve, pl.: 994 Scarbantia Intercity (String)
- `vonatTipus:` vonat típusa, pl.: Intercity, Railjet, Személyvonat (string)
- `allomasok` vonat állomásai egy Allomas* tömbben (Allomas*)
- `kocsik` vonat kocsijai (Kocsi*)
- `idok` milyen időközönként indulnak a vonatok (percben) (Ido)

A vonatokhoz tartozó kocsik szintén el lesznek tárolva egy külön „.txt” fájlban az alább látható módon:

sorSzam;ferohely;kerekpar

Paraméterek – zárójelben jelölve, milyen változóval lesz eltárolva a programban

- `sorSzam` kocsi sorszáma (int)
- `ferohely` kocsi férőhelye (int)
- `kerekpar` kocsiban van kerékpártároló? (bool)

Az alábbi paraméterek az állomáshoz szólnak, az alábbi módon lesz eltárolva hasonló módon „.txt” fájlban:

allomasNev;vonatMegall;elozoAllomas;kovAllomas

Paraméterek – zárójelben jelölve, milyen változóval lesz eltárolva a programban

- `allomasNev:` az állomás neve (String)
- `vonatMegall`: vonatok, amik ezen állomáson megállnak (Vonat*)
- `elozoAllomas`: előző állomás (String*)
- `kovAllomas`: következő állomás (String*)

Ehhez kapcsolódik a menetrend osztály, amiben csak egyetlen változó lesz egy sorban, az „elsoAllomas”, ami megadja a láncolt lista kezdőpontját

Az alábbi paraméterek az Ido osztályhoz szólnak, kivételesen ezek nem lesznek eltárolva „.txt” fájlban.

Paraméterek – zárójelben jelölve, milyen típusúak az attribútumok:

- `ev` (int)
- `honap` (int)
- `nap` (int)
- `ora` (int)
- `perc` (int)

Az alábbi paraméterek a String osztályhoz szólnak, ezek sem lesznek eltárolva „.txt” fájlban.

Paraméterek – zárójelben jelölve, milyen típusúak az attribútumok:

- `pData:` a String karaktertömbje (char*)
- `len:` a String hossza ’\0’ lezáró jel nélkül (size_t)

Egyéb és fontosabb algoritmusok

A menü switch-case-zel lesz megoldva a main() függvényben. Lépkedni az opciók között billentyűkkel lehet, a switch egyes case-eiben fogják megjeleníteni az új menüpontokat. Ez a menü egy while ciklusban lesz megvalósítva, és addig fog futni, ameddig a felhasználó le nem nyomja az adott billentyűt.

A program kezdetekor beolvassa a kívánt fájlokat, ezekhez lesz külön-külön, ilyen célra lementett „.txt” kiterjesztésű fájlok, ahonnan be lesz olvasva. Ezeket láncolt listákban fogja tárolni.

Vonat függvényei

A konstruktor függvényekkel természetesen az osztály példányosítását lehet megvalósítani. Van egy paraméter nélküli konstruktor, és van egy, amiben az összes paraméter van.

void kocsiAdd() ezzel a függvénnyel lehet beszúrni egy új kocsit sorszám alapján

Kocsi függvényei

A konstruktor függvényekkel természetesen az osztály példányosítását lehet megvalósítani. Van egy paraméter nélküli konstruktor, és van egy, amiben az összes paraméter van.

void listazas(os: ostream&)

kiírja az objektum adatait tördelten

Jegy függvényei

A konstruktor függvényekkel természetesen az osztály példányosítását lehet megvalósítani. Van egy paraméter nélküli konstruktor, és van egy, amiben szinte összes paraméter van, azért, mivel a felhasználó nem írhatja be magának a jegy árát, sem a teljes távot, sem az ülőhelyét, hanem ezt megcsinálja helyette a program.

void adatKiir(os: ostream&)

kiírja az objektum adatait a konzolra tördelten

int teljesT(indulas, erkezes)

kiszámolja a teljes távját az útnak, visszaadja a teljesTav-ba ezt km-ben

int arKiszamolasa() ez a függvény kiszámítja a jegy árát figyelembe véve a kedvezmények és a vonat típusát

String ulohelyGen() ez a függvény generálja a jegyhez tartozó ülőhelyet

void kesesKezeles() ez a függvény random generálja, hogy késik-e a vonat vagy sem, és ha igen, akkor mennyit késik.

Menetrend függvényei

A konstruktor függvényekkel természetesen az osztály példányosítását lehet megvalósítani. Van egy paraméter nélküli konstruktor, és van egy, amiben az összes paraméter van.

void listazas(os: ostream&)

kiírja a láncolt lista elemeit

void allomasAdd() a láncolt listához hozzáad még egy elemet

void allomasDel() a láncolt listából kitöröl egy elemet

Allomas allomasKeres(String all)

a listában megkeresi a paraméterül kapott állomást

Ido függvényei

A konstruktor függvényekkel természetesen az osztály példányosítását lehet megvalósítani. Van egy paraméter nélküli konstruktor, van egy két paraméteres konstruktor, amivel az óra;perc szerint lehet példányosítani, illetve van egy három paraméteres, amivel év.hónap.nap formátumban lehet példányosítani.

void kiirTeljes() ez a függvény kiírja rendre az egész objektumot

void kiirOraPerc() ez a függvény az óra és a perc attribútumot írja ki

int getNapokAHonapban(int, int)

ez a függvény visszaadja az adott hónap napjainak számát, ügyelve a szökőévekre

Ido operator-(const Ido&)

ez a függvény lehetővé teszi, hogy két időt ki lehessen vonni egymásból

String függvényei

A konstruktor függvényekkel természetesen az osztály példányosítását lehet megvalósítani. Van egy, ami egy nullával lezárt char sorozat a paramétere, van egy char paraméteres konstruktor, amivel egy char karakterből lehet példányosítani, illetve van egy másoló konstruktor, ami egy String objektumot kér paraméternek.

String& operator=() ez a függvény overloadolja az „=” operatort

String operator+() ez a függvény overloadolja a „+” operatort, két Stringet összefűz

String operator+(char) ez a függvény overloadolja a „+” operatort, egy karaktert fűz a Stringhez

char& operator[]() ez a függvény overloadolja az „[]” operatort, egy megadott indexű elemének referenciájával tér vissza

const char& operator[]()

ez a függvény overloadolja az „[]” operatort, egy megadott indexű elemének referenciájával tér vissza, viszont ez const adattagra érvényes

std::ostream& operator<<()

ez a függvény kiír az ostream-re

std::istream& operator>>()

ez a függvény az istream-ről egy szót beolvas egy Stringbe

inline String operator+()

ez a függvény karakterhez Stringet fűz

Tárolás

E célból született meg a „TemplateKontener” osztály. Ez az osztály sablonosan tárolja el a különböző adatokat, duplán láncolt listával. Először is, ebben található egy struktúra, ami magát a lista elemeit definiálja. Minden lista elemnek van egy „T” adata, illetve két pointer, ami az előző, illetve a következő elemre mutat. Ezután ennek az osztálynak az adattagjai egy „head” és egy „tail”. A „head” attribútum a lista „feje”, azaz a legelső elemre mutató pointer, a „tail”, vagyis a „farok” meg az utolsó elemre mutató pointer. A „size” egy egész szám, ami tárolja a lista méretét, vagyis a benne található elemek számát. Ezzel ki lehet küszöbölni, hogy minden egyes osztályhoz külön-külön létrehozzunk új tároló osztályokat. Az osztály implementálja a listához szükséges alapvető műveleteket, mint például az elemek hozzáadását a lista végéhez, elemek hozzáadását a lista elejéhez, elemek eltávolítását a lista végéről, elemek eltávolítását a lista elejéről, lista ürességének ellenőrzését, lista méretének lekérdezését és lista kiürítését.

A TemplateKontener-nek egy konstruktora van, ami paraméter nélküli, természetesen a „head” és a „tail” pointereket null pointernek hozza létre, a „size” pedig alapértelmezetten 0.

void vegeBeszur(const T& value)

Ez a függvény a létrehoz egy új Lista elemet, kezeli azt, hogy üres-e vagy sem, ha igen, akkor a „head” és a „tail” is az új Lista elem lesz, különben a „tail” erre az új Lista elemre fog mutatni.

void elejeBeszur(const T& value)

Ez a függvény a létrehoz egy új Lista elemet, kezeli azt, hogy üres-e vagy sem, ha igen, akkor a „head” és a „tail” is az új Lista elem lesz, különben a „head” erre az új Lista elemre fog mutatni.

void kozepereBeszur(const T& elozo, const T& value)

Ez a függvény a paraméterül kapott „elozo” Lista elem után szúrja be a szintén paraméterül kapott „value” Lista elemet.

void torlesVegerol()	Ez a függvény a Lista végét törli, az előző Lista elemet teszi „tail”-é, és csökkenti eggyel a Lista méretét.
void torlesElejerol()	Ez a függvény a Lista elejét törli, az előző Lista elemet teszi „head”-dé, és csökkenti eggyel a Lista méretét.
void torlesParam()	Ez a függvény a paraméterül kapott elemet törli a Listából, ha a „head” vagy a „tail” elem az, akkor meghívja az ezekhez kapcsolódó 2 függvényt.
bool isEmpty() const	Ez a függvény lekérdezi, hogy az adott Lista üres-e vagy sem, és ezt adja vissza.
int getSize() const	Ez a függvény visszaadja a Lista méretét.
void clear()	Ez a függvény teljes egészében törli a listát.

void print(const T&)	Ez a függvény kiírja tördelten a konzolra a paraméterül kapott Lista elemet