Nagy házi feladat terv

Telefonközpont szimuláció

Benedek Katalin
XXCAYJ
Labor vezető: Gera Dóra

Felhasznált eszközök, programozási nyelvek:

- Programozási nyelvek:
  - C++
- Operációs rendszer:
  - Windows 10
- Programok:
  - CodeBlocks
- Egyéb:
  - memtrace – memóriaszivárgás elkerülése végett
  - g_test_lite – tesztesetek szimulálása

A feladat:

Készítsen egyszerű telefonhálózatot szimuláló **előfizető** és **központ** osztályokat!

- A központtól egy előfizető vagy egy másik központ kapcsolást kérhet. Az egyes előfizetőket 4 jegyű hívószámmal lehet elérni, melyből az eső jegy a körzetszám (1-9), a maradék 3 jegy pedig az előfizetői szám.
- Minden körzetben pontosan egy központ van. Ehhez kapcsolódnak a körzet előfizetői és az idegen körzetek központjai. Két központ között több kapcsolat is lehet.
- A központok külső kapcsolata a létrehozásakor megadott maximumot nem haladhatja meg. Hasonlóan a központok létrehozásakor adható meg, hogy az adott központ hány kapcsolási kérést tud egyszerre kiszolgálni (kapcsolási tábla mérete).
- Ha egy előfizető kapcsolást kér, meg kell próbálni a kapcsolatot fölépíteni. Ezt tárolni kell a korlátos méretű kapcsolótáblában.
- Távolsági hívásoknál minden érintett központban egy-egy kapcsolás keletkezik. Ha valamelyik központ nem tudja teljesíteni a kapcsolást, mert nincs szabad hely a kapcsolótáblában, nincs szabad távolsági vonal, vagy foglalt az előfizető, akkor kivételt dob, minek hatására törlődik az addig felépített kapcsolási sor. Ellenkező esetben a hívó egy kapcsolás-objektumot kap, amelyen keresztül üzenetet küldhet. A hívó megszakíthatja a hívást, ekkor törlődik a kapcsolás, és erről értesül a hívott fél is.
- Demonstrálja a működést külön modulként fordított tesztprogrammal! A megoldáshoz felhasználhat STL tárolót is!

Pontosított specifikáció (megegyezik az eredetivel):

A program egy parancssori alkalmazás, melynek célja, egy telefonközpont felépítésének és működésének modellezése.
Az alkalmazás megfelelő működéséhez szükséges kezdeti ***bemeneti adatok*** a következőek:

- **Körzet objektumok listája**: Ez adja meg a hálózat felépítését, leírja a körzeteket, központokat és a köztük lévő kapcsolatokat is. Továbbá tartalmazza a körzetekhez tartozó egy-egy központ objektumot is (kapcsolótábla, kapcsolótáblák maximális mérete… stb.) Enélkül nem jöhetne létre semmilyen kommunikáció. Ez adja meg a modell alapját, nélkülözhetetlen a működés szempontjából, ezért kötelező adat. A lista maximum 9 db körzetet tartalmazhat.
- **Előfizetők**: A kommunikációban részt vehető feleket leíró adat. A kapcsolatteremtés kezdeményezéséhez elengedhetetlen, azonban a program enélkül is lefut. A funkciók nagy része nem használható ezen bemenet hiányában. A körzetek nem tartalmazzák ezt az adatot, a kapcsolatot az előfizetők azonosítói adják meg.
- **Egyéb:** A program működése során kérhet be a felhasználótól egy-egy adatot, hogy a szimuláció minél pontosabb legyen. A bemenet várt formátuma ezen esetekben (akár példával illusztrálva is) megjelenik a képernyőn. (Ilyen pl.: Ki a hívó fél? → pl: XYZ)

Alapértelmezetten ezeket a program a megfelelő fájlból olvassa be, de a kezdeti adatok a programon belül, később is bővíthetők, de akkor már csak egyesével. Módosításuk nem lehetséges, ezért törölni, majd újra létrehozni kell hiba esetén.

A bemenő adatok beolvasásakor a megfelelő formátumról a felhasználónak kell gondoskodnia, amennyiben egy adott adat nem helyes formátumban van, azt a program nem tudja feldolgozni, ami adott esetben hibás működést eredményezhet. (Ennek formai pontosítása a későbbiekben történik majd).
Ha a bemeneti adatok mindegyike kielégítő, a következő ***funkciók*** érhetőek el a felhasználó számára:

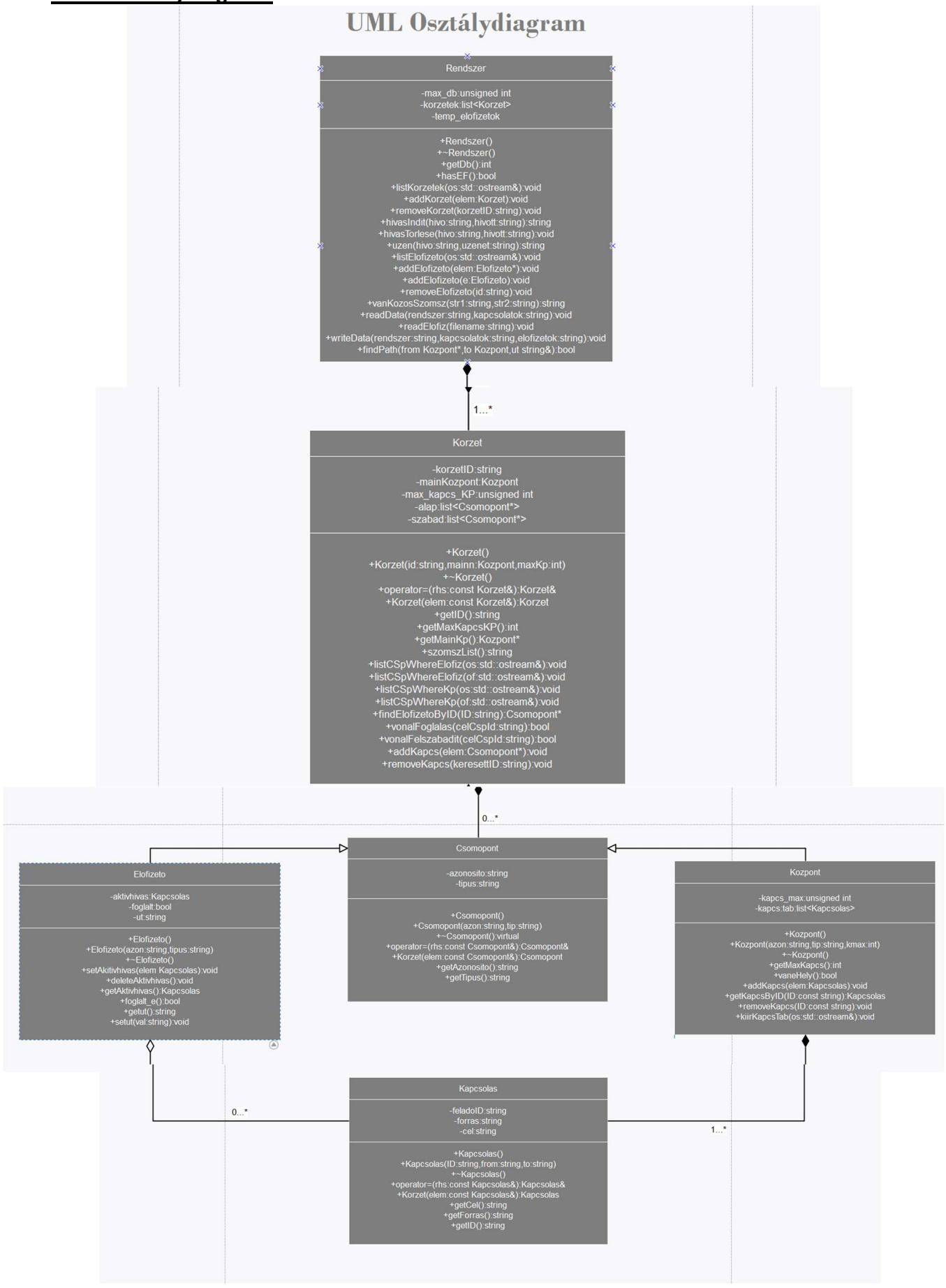
- **Körzetek kezelése**: Új körzet hozzáadása, régiek törlése, listázás.
- **Előfizetők kezelése**: Új előfizető hozzáadása, -törlése, listázás.
- **Hívás indítása**: Két fél között kiépíthető egy kommunikációs csatorna, ahol megszakításig üzenetek küldhetőek.
- **Üzenet küldése:** Meglévő kapcsolat esetén a hívó fél üzeneteket küldhet a hívott félnek. Az üzenetek nem visszaolvashatóak, nem kerülnek mentésre. (Ez a funkció egyoldalú, azaz válaszhoz előbb bontani kell a kapcsolatot, majd újabb hívást kezdeményezni a másik irányból)
- **Hívás befejezése**: Meglévő kapcsolat lezárása. Ez a funkció üzenetet küld a lezárás előtt a hívott félnek a hívás befejezéséről.

A program által generált file-ok és ***kimenetek***:

- **Körzetek/Előfizetők listája**: Szöveges fájlba és esetlenként képernyőre kiírt adat. Az újonnan bevitt adatokat is ebbe menti el. Törlések esetén is frissül.
- **Sikeres/Sikertelen hívás:** Hívás indítás esetén a képernyőn megjelenő üzenet tájékoztatja a felhasználót arról, hogy sikerrel járt-e. Ha nem mi volt az oka.
- **Kapcsolási útvonal:** Képernyőre kiírt adat. Az üzenetküldés menüpont alatt jelenik meg. Nem kerül külön mentésre.
- **Egyéb:** A képernyőn megjelenhetnek kisebb jelentőségű üzenetek, amik azt a célt szolgálják, hogy a felhasználó minél könnyebben használja a programot. Ezek egyetlen esetben sem kerülnek mentésre.

A program csak a megjelölt esetekben menti a modellt, így bizonyos adatok a szimuláció bezárásával elvesznek. (Pl.: Kiépített kapcsolási útvonalak, aktív hívások.)

Terv - Osztálydiagram:



Megoldás vázlata:

A megoldás során top-down tervezést alkalmaztam. A célom az volt, hogy mind a teszteseteket és a felhasználó által használható programot is elkészítsem. Először megterveztem a bementeket, majd a felhasználói tesztet. Aztán szép sorban az összes osztályt elkészítettem. Amikor minden tagfüggvény elkészült kiegészítettem a tesztjeimet velük, és a felmerülő problémákat kijavítottam. Ezután elkészítettem a g_test_lite program segítségével a gépi tesztet, amelyhez az alapot a felhasználó teszt nyújtotta. Ezek a teljes program 76,16% -át fedik. Ezután felhasználtam a memtrace file-t, hogy teszteljem, nincs-e memóriaszivárgás a kódban. Végül kommentekkel egészítettem ki a megoldásomat, hogy könnyebben átlátható legyen.

Osztályok→ Rendszer:

Leírás:

A főbb funkciók ellátását és a komplett rendszermodellt ez az osztály tartalmazza. Melynek körzetek tömbje (feladatban meghatározott adatok alapján) maximum 9 db elemet tartalmazhat. Ezek egyenként a **Korzet** osztály objektumai.

Függvényei:

```
///Kiirja a korzeteket es minden lenyeges tualajdonsagukat
void Rendszer::listKorzetek(std::ostream& os){
///Ha nem ertuk el a maximalis meretet akkor hozzaadunk egy uj korzetet
void Rendszer::addKorzet(Korzet elem) {
///Id alapjan toroljuk a korzetet a listabol
void Rendszer::removeKorzet(string korzetID) {
///Listazza a kapcsolt elofizetoket
void Rendszer::listElofizeto(std::ostream& os){
///Hozzaad egy uj elofizetot -filebol
void Rendszer::addElofizeto(Elofizeto* elem) {
///Hozzaad egy uj elofizetot - felhasznalotol
void Rendszer::addElofizeto(Elofizeto e) {
///ID alapjan torol egy elofizetot
void Rendszer::removeElofizeto(string ID) {
///Megszakit egy hivast A-bol B-be
void Rendszer::hivasTorlese(string hivo,string hivott) {
///Ha letezik a kapcsolat akkor uzenetet kuld A-bol B-be - ketiranyusitva
string Rendszer::uzen(string hivo, string uzenet) {
///Megadja hogy ket kozpontnak van-e kozos szomszedja, azaz hogy max 2 hosszu uton el lehet jutni A-bol B-be
string Rendszer::vanKozosSzomsz(string str1, string str2){
///Megadja hogy van-e ut A-bol B-be, ha van fogalal is
bool Rendszer::findPath(Kozpont* from, Kozpont* to,string& ut){
///Ha lehetseges hivast indit, ha nem jelzi mi volt a baj
string Rendszer::hivasIndit(string hivo, string hivott){
///A harom parameterben kapott fileba kiirja azokat az adatokat, amiket be is tudunk olvasni
void Rendszer::writeData(string rendszer,string kapcsolatok,string elofizetok){
///Korzetek es kozpontokat tartalmazo filetokat olvas es alakitja ki belole, amit kell. Elofizetok hozzaadasa nem itt tortenik.
///Elso parameter: maga a rendszer; Masodik: a koztuk levo kapcsolatok
void Rendszer::readData(string rendszer,string kapcsolatok){
///Beolvassa es a megfelelo helyre menti az elofizetoket a parameterben megadott filebol
void Rendszer::readElofiz(string filename){
 ///Visszaadja hogy jelenleg hany korzet van
 int getDb(){return korzetek.size();}
  ///Visszaadja hogy van-e elofizeto
 bool hasEF(){return (temp_elofizetok.size()!=0);}
```

Osztályok→ Korzet:

Leírás:

Az osztály főként a nagyobb funkciókhoz szükséges kisebb műveletek elvégzésére alkalmas függvényeket tartalmaz. Legfontosabb adattagja egy heterogén kollekció, amelynek alapját a Csomopont osztály adja.

Függvényei:

```
///Az elofizetok kilistazasara hasznalt fuggveny - ostream-re
void Korzet::listCspWhereElofiz(std::ostream& os){
///Az elofizetok kilistazasara hasznalt fuggveny - file-ba
void Korzet::listCspWhereElofiz(std::ofstream& of){
///Az kozpontok kilistazasara hasznalt fuggveny - ostream-re
void Korzet::listCspWhereKp(std::ostream& os){
///Az kozpontok kilistazasara hasznalt fuggveny -file-ba
void Korzet::listCspWhereKp(std::ofstream& of){
///Visszaad egy csomopont elemet id alapjan
Csomopont* Korzet::findElofizetoByID(string ID){
///Visszaadjuk egy stringben egy korzet osszes kapcsolodo korzetet
string Korzet::szomszList(){
///A megadott helyre vonalat foglal. Azaz ha a szabad vonalak kozott van olyan, aminek az id-ja megegyezik a megadottal, akkor azt kitoroljuk onnan.
///Visszaadjuk, hogy sikerult-e.
bool Korzet::vonalFogalalas(string celCspId){
///A megadott helyre vonalat szabadit fel. Azaz ha az alap vonalak kozott van olyan, aminek az id-ja megegyezik a megadottal, akkor azt hozzaadjuk a szabad vonalakhoz.
///Visszaadjuk, hogy sikerult-e.
bool Korzet::vonalFelszabadit(string celCspId){
///A korzethez hozzaad egy uj elofizetot, vagy egy kozpont kapcsolatot (ennel kikotes, hogy ne lepjuk tul a maximalis kapcsolatok szamat)
void Korzet::addKapcs(Csomopont* elem){
///A korzetbol kitorol egy elofizetot, vagy egy kozpontot
void Korzet::removeKapcs(string keresettID){
///Visszaadja a korzet azonositot
string getID(){return korzetID;}
///Visszaadja a maximalis kiepitheto kapcsolatok szamat
int getMaxKapcsKp(){return max_kapcs_KP;}
///Visszaad egy pointert a korzet sajat kozpontjara
Kozpont* getMainKp() {return &mainKozpont;}
```

Osztályok→ Csomopont:

Leírás:

Ez egy ősosztály, mely jelenleg két különböző származtatott osztályt tartalmaz, de az igény szerint bővíthető bármely programozó által.

Függvényei:

```
///Visszadja a csomopont azonositojat
string getAzonosito(){return azonosito;}
///Visszaadja a csomopont tipusat
string getTipus(){return tipus;}
```

Osztályok→ Kozpont/ Elofizeto:

Leírás:

A meglévő csomópont típusok az Elofizeto és a Kozpont. Mindkét osztály tartalmaz egy adattagot, amely vagy egy, vagy több Kapcsolas objektumot tartalmaz.

Függvényei:

```
///Beallitja az aktiv hivast egy Kapcsolasra es a foglaltsagot is
void setAktivhivas(Kapcsolas elem){ aktivhivas=elem; foglalt=true;}
///Feloldja a foglalatsagot
void deleteAktivhivas(){foglalt=false;}
///Visszaadja az aktiv hivas tulajdonsagait
Kapcsolas getAktivhivas(){return aktivhivas;}
///Megadja hogy az Elofizeto foglalt-e
bool foglalt_e(){return foglalt;}
///Visszaadja az utat a hivashoz
string getut(){return ut;}
///Beallitja az utat a hivashoz
void setut(string val){ut=val;}
```

```
///Visszaadja, hogy van-e meg hely a kapcsolotablaban
bool Kozpont::vaneHely(){
///Ha van hely, hozzaadja a parameterben megadott elemet a kapcsolotablahoz
void Kozpont::addKapcs(Kapcsolas elem){
///Visszaad egy Kapcsolas elemet a parameterben megadott feladoID szerint. Ha nincs ilyen hibat dob
Kapcsolas Kozpont::getKapcsByID(const string ID){
///A parameterben megadott feladoID alapjan kitorli az elemet a kapcsolotablabol
void Kozpont::removeKapcs(string ID){
///Kapcsolotabla kiiro fuggveny
void Kozpont::kiirKapcsTab(std::ostream& os){
///Visszaadja a maximalis kapcsolasok szamat amit a kapcsolotabla tud kezelni
int getMaxKapcs(){return kapcs_max;}
```

Osztályok→ Kapcsolas:

Leírás:

Az osztály egy adatstruktúra csak, különösebb funkciója nincs.

Függvényei:

```
///Visszaadja a hivas celjat
string getCel(){return cel;}
///Visszaadja a hivas forrasat
string getForras(){return forras;}
///Visszaadja a hivas kitol indult
string getID(){return feladoID;}
```

Tesztelés – Teszt program:

A teszt program külön állományként futtatható. Ha a CPORTA makró definiálva van, akkor az automatikus tesztek futnak le. Ezekhez a gtest_lite modult használtam fel. Ha a makró nem létezik, akkor a felhasználói interfész jelenik meg, ami a console ablak. Itt egy menü segítségével lehet elérni a különböző funkciókat. Az automata rész ezeket a funkciókat szimulálja le.

Tesztek:

- INPUT TEST: Kezdeti adatok beolvasása.
- OUTPUT TEST: Az adatok új fájlba írás is működik-e.
- KORZET TEST: Új hozzáadása, listázás, törlés
- ELOFIZETO TEST: Új hozzáadása, listázás, törlés
- HIVAS TEST: Jó es hibás tesztek egyaránt
- UZEN TEST: Jó üzenet, rossz üzenet egyaránt
- HIVAS LERAK TEST: megpróbálja lerakni a meglévő hívások valamelyikét

Tesztelés – Memória szivárgás:

A memóriakezelést a laborokon is használt MEMTRACE modullal ellenőriztem. Minden forrásfájlba beillesztettem. A futás során egyik teszt/menüpont futása sem okozott memóriaszivárgást. A Jporta rendszerbe való feltöltés után sem mutatott a programom memóriakezelési problémát.

Bemenetek/Kimenetek:

- Rendszer input: (korzetek_be.txt) Felhasználói tesztben lehet máshonnan olvasni.
- Kapcsolatok input: (kapcsolatok_be.txt) Felhasználói tesztben lehet máshonnan olvasni.
- Előfizetők input: (elofizetok_be.txt) Felhasználói tesztben lehet máshonnan olvasni.

- Rendszer output: (korzetek_def_ki.txt) Ide mindenképp ment. Ez az alap mentés minden módosítás után.
- Kapcsolatok output:(kapcsolatok_def_ki.txt) Ide mindenképp ment. Ez az alap mentés minden módosítás után.
- Előfizetők output: (elofizetok_def_ki.txt) Ide mindenképp ment. Ez az alap mentés minden módosítás után.

- Rendszer output: (korzetek_uni.txt) Felhasználói tesztben lehet máshova menteni.
- Kapcsolatok output:(kapcsolatok_uni.txt) Felhasználói tesztben lehet máshova menteni.
- Előfizetők output: (elofizetok_uni.txt) Felhasználói tesztben lehet máshova menteni.

Dátum: Budapest, 2020.05.17.

Benedek Katalin

Benedek Katalin