

# **Telefonkönyv specifikáció**

Programozás alapjai 2

Kilián Marcell

V13FLM

2023. április 16.

## Feladat

Tervezze meg egy telefonkönyv alkalmazás egyszerűsített objektummodelljét, majd valósítsa azt meg! A telefonkönyvben kezdetben az alábbi adatokat akarjuk tárolni, de később bővíteni akarunk:

- név (vezetéknév, keresztnév)
- becenév
- cím
- munkahelyi szám
- privát szám

Az alkalmazással minimum a következő műveleteket kívánjuk elvégezni:

- adatok felvétele
- adatok törlése
- listázás

A rendszer lehet bővebb funkcionalitású (pl. módosítás, keresés), ezért nagyon fontos, hogy jól határozza meg az objektumokat és azok felelősségét. Demonstrálja a működést külön modulként fordított tesztprogrammal! A megoldáshoz ne használjon STL tárolót!

---

## Program célja

Egy konzolos telefonkönyv megvalósítása, ahol az emberek adatait lehet kezelni. A program képes ezeknek az adatoknak a listázására, új adat felvételére, meglévő adat törlésére, keresésére és egy adat frissítésére.

Kezdetben a telefonkönyv az emberek nevét, becenevét, címét, munkahelyi vagy privát számát tárolja, ezeket az alap adatokat később bővíteni lehet más, a felhasználó által megadott adatokkal. Minden adat egy szóból állhat, egyik adat sem tárol el space karaktert.

---

## Program használata

A program indításakor a menüben találja magát a felhasználó, itt dönthet, hogy milyen műveletet szeretne csinálni

- új telefonkönyv létrehozása
- telefonkönyv megnyitása

Ha az új telefonkönyv **létrehozását** választotta a felhasználó, választhat, hogy az alap adatokkal szeretné létrehozni egy telefonkönyvet, vagy szeretne-e további adatokat hozzáadni a telefonkönyvéhez. Ezek után van lehetősége a felhasználónak adatok felvételére, listázására, törlésére, frissítésére. Ha a telefonkönyv **megnyitását** választotta a felhasználó, a program kéri a felhasználótól a fájl nevét, amiből a program betölti a telefonkönyv adatait, egy létező telefonkönyv importálása után, lehetősége van a telefonkönyv adatait megjeleníteni, bővíteni adatokkal, hozzáadni új adatot, törölni és frissíteni a már meglévő adatokat.

A telefonkönyv **listázását** választva, az emberek nevei, ABC szerint, növekvő sorrendben listázódnak, jelennek meg. A telefonkönyvben szereplő emberek adatai egymás alatt láthatók, egy ember adatai között egy tabulátorni hely van.

---

Ha **új adatot** szeretne felvenni a telefonkönyvbe a felhasználó, az alapvető adatokat ki kell töltenie, az ember nevének egyedinek kell lenniük. Egy adat felvételénél a program egyesével megkérdezi, hogy mi az embernek a neve, beceneve, címe és a többi adata, az alap adatokon felüli adatok lehetnek üresek, nem kötelező kitölteni ezeket az adatokat.

---

Mivel az emberek nevei egyediek, ezért a **törlés** és a **keresés** az emberek nevének megadásával történik. A törlés lehetőség kiválasztása után a felhasználó azt az adatot törli a telefonkönyvből, akinek a nevét adja meg. A keresés lehetőség kiválasztása után, a törléshez hasonlóan, a név megadása után, ha létező nevet ad meg a felhasználó, megjelennek az ember adatai.

---

Az **adatok frissítése, módosítása** hasonlóan történik a kereséshez és az új adat felvételéhez, miután megtaláltuk a keresett emberünket, megjelenítjük egyesével, ahogy az új adat hozzáadásánál is, az alap, illetve hozzáadott adatokat (név, becenév stb.). Ha a felhasználó sima – (kötőjel) karaktert ad meg az egyik adatnál, akkor azt az adatot nem változtatja meg a program. Ha eltérő adatot ad meg a felhasználó a jelenlegitől, az adat frissül.

---

## Tesztelés

Kritériumok, amikre a program figyel:

- Az ember nevében, becenevében nem szerepelhet szám karakter
- Az ember címében legalább 1 betű és szám karakternek kell szerepelnie
- A telefonszámokban csak + jel és számok lehetnek
- Minden ember nevének egyedinek kell lennie (szerkesztésnél is)

A program tesztéseinek elkészítéséhez a gtest\_lite.h fájlt használtam, a tesztéseket a teszt.h fájlban készítettem el. (Mivel az osztályok több metódusait is le lehet egyben tesztelni, ezeket a tesztéseket összevontam egy tesztbe. A teszt.h-ban lévő tesztek sikeresen lefutottak, azt az eredményt kaptam a teszteléseknél amiket vártam.)

A memóriaszivárgást a memtrace.h+.cpp állományok használatával ellenőrzöm.

Tesztelések a teszt.h fájlban:

TEST(Vektor, Add Remove Getterek) – A Vektor osztály Add, Remove és getter metódusait

TEST(Vektor, operator[]) – A Vektor osztály [] operátorát teszteli

TEST(String, Konstruktorok Getterek) – A String osztály konstruktorait és getter metódusait

TEST(String, operator[]) – A String osztály [] operátorát teszteli

TEST(String, VanESzam lehetETelefonszam) – A VanESzam és lehetETelefonszam metódusokat teszteli

TEST(String, Operator<<) – A String osztály << operátorát teszteli

TEST(String, Operator=) – A String osztály = operátorát teszteli

TEST(String, Operator+=) – A String osztály += operátorát teszteli

TEST(StringPar, Konstruktorok Getterek) – A StringPar osztály konstruktorait és gettereit

TEST(StringPar, operator== Setterek) – A StringPar osztály == operátorát és settereit teszteli

TEST(Telefonszam, Konstruktorok Getterek Setterek operator=) – A Telefonszam osztály konstruktorait, gettereit, settereit és az = operátorát teszteli

TEST(Ember, Konstruktorok Getterek Setterek operator=) – Az Dolgozo és Maganember osztályok konstruktorait, gettereit, settereit és = operátor metódusait teszteli

TEST(Telefonkonyv, Konstruktor Getter Setter) – A Telefonkonyv osztály konstruktorát, gettereit és settereit teszteli

---

## Fontosabb metódusok

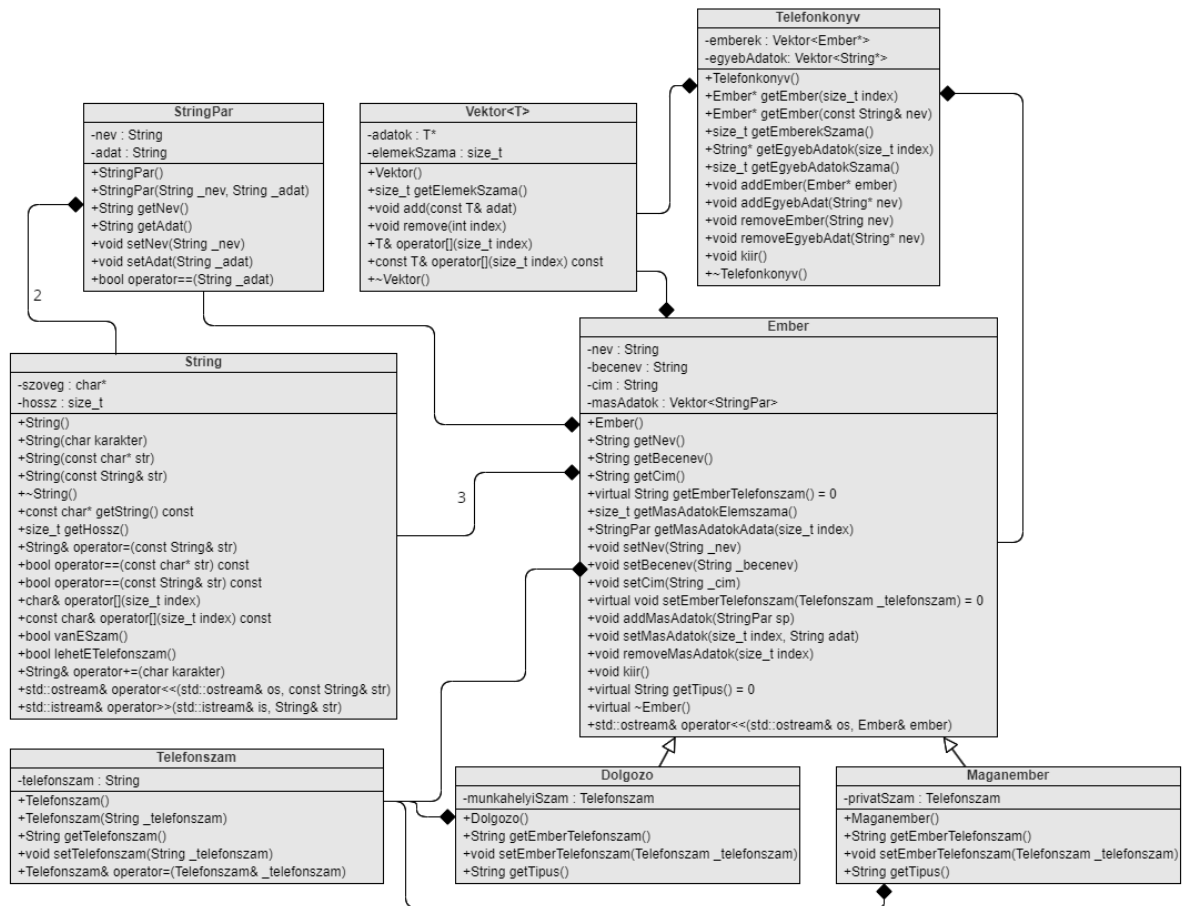
**void menu()** //Innen indul a program, betölti a menüt, megjeleníti, hogy milyen opciói vannak a felhasználónak (telefonkönyv létrehozása vagy betöltése)

**void telefonkonyv\_muveletek(Telefonkonyv t)** //A telefonkönyv betöltése vagy létrehozása után ez a metódus irányítja a felhasználót a telefonkönyvvel kapcsolatos műveletekhez, itt döntheti el a felhasználó, hogy mire szeretné a telefonkönyvet használni

**void telefonkonyv\_mentes(Telefonkonyv t)** //Elmenti egy txt fájlba a telefonkönyvet, később betölthetjük a telefonkönyvet a programban

**void telefonkonyv\_betolt()** //Betölt egy elmentett telefonkönyvet, figyel, hogy az első sor beolvasásánál az alap adatokon kívül eltárolja a többi/egyéni adatokat is

# Osztálydiagram



## Osztályok és a hozzá tartozó metódusok

### String //Karaktersorozatok kezeléséhez

adattagok:

char\* szoveg //A karaktersorozat itt tárolódik

size\_t hossz //A karaktersorozat hossza (\0-t nem számoljuk bele)

metódusok:

String() //Egy üres karaktersorozatot hozunk létre

String(const char\* str) //Egy bejövő karaktersorozatot eltárolunk

String(const String& str) //Egy másik String-ből másolunk

~String() //Felszabadítja a lefoglalt szoveg-et

const char\* getString() const //Visszaadja a szoveg kezdő pointerét

size\_t getHossz() //Visszaadja a String hossz értékét

String& operator=(const String& str) //Lemásolja a String-et

bool operator==(const char\* str) const //Megvizsgálja, hogy a Stringek szöveg adatai megegyeznek-e bejövő szöveges paraméterrel

bool operator==(const String& str) const //Megvizsgálja, hogy a Stringek szöveg adatai megegyeznek-e bejövő String paraméterrel

char& operator[](size\_t index) //Visszaadja az indexedik elemét a karaktertömbnek

const char& operator[](size\_t index) const //Visszaadja az indexedik elemét a karaktertömbnek (const)

bool vanESzam() //Megmondja, hogy szerepel-e szám a String-ben

bool lehetETelefonszam() //Megmondja, hogy telefonszám lehet-e a String

String& operator+=(char karakter) //A String-hez hozzáad egy karaktert a szöveg végére

std::ostream& operator<<(std::ostream& os, const String& str) //Kiírásoknál a szöveget kell megjeleníteni

std::istream& operator>>(std::istream& is, String& str) //String-be beolvashatunk input adatot

**StringPar** //Egy extra adat megnevezését és értékét tároljuk itt

adattagok:

String nev //A további adat megnevezése

String adat //A további adat értéke

metódusok:

StringPar() //Egy üres nevet és adat párt hoz létre

StringPar(String \_nev, String \_adat) //Egy \_nev nevet és \_adat adatot hoz létre

String getNev() //Visszaadja a nevét az extra adatnak

String getAdat() //Visszaadja az értékét az extra adatnak

void setNev(String \_nev) //Beállíthatunk egy új értéket a névnek

void setAdat(String \_adat) //Beállíthatunk egy új értéket az adatnak

bool operator==(String \_adat) //Megmondja, hogy megegyeznek-e a két adat érték

**Vektor** //Dinamikus memóriakezeléshez használjuk

adattagok:

T\* adatok //Egy T tömböt tárol

size\_t elemekSzama //A T tömb elemszámát tárolja

metódusok:

```
Vektor() //Létrehoz egy üres tömböt, 0 elemszámmal  
size_t getElemekSzama() //Az elemek számát adja vissza  
void add(const T& adat) //Egy T elemet adhatunk a tömbünkhöz  
void remove(size_t index) //Egy elemet töröl a tömbünkből  
T& operator[](size_t index) //A tömb egyik elemével tér vissza  
const T& operator[](size_t index) const //A tömb egyik elemével tér vissza (const)  
~Vektor() //Felszabadítja a dinamikusan foglalt adatok memóriát
```

**Telefonszam** //telefonszámok létrehozásához

adattagok:

```
String telefonszam //A telefonszámot tároljuk itt
```

metódusok:

```
Telefonszam() //Egy üres Telefonszam-ot hoz létre  
Telefonszam(String _telefonszam) //Egy telefonszámot hoz létre a bejövő String paraméterrel  
String getTelefonszam() //Visszaadja a telefonszámot  
void setTelefonszam(String _telefonszam) //A telefonszámnak egy új értéket lehet beállítani, figyelni kell, hogy egyedi maradjon a telefonszám  
Telefonszam& operator=(Telefonszam& _telefonszam) //Lehet, hogy félreírtunk egy telefonszámot, a javításához kell
```

**Ember** //Ember adatainak tárolásához

adattagok:

```
String nev //Az ember nevét tárolja, az ember neve egy szóból állhat  
String becenev //Becenevét  
String cim //Címét  
Vektor<StringPar*> masAdatok //Az egyéb adatokat, amiket hozzáadtunk itt tároljuk
```

metódusok:

```
Ember() //A sima konstruktor egyesével bekéri a felhasználótól az Ember paramétereit, az alapvető adatokat ki kell tölteni, az adatokat egyesével a set metódusok segítségével állítjuk majd be
```

```
String getNev() //Az ember nevét adja vissza
```

```

String getBecenev() //Az ember becenevét adja vissza

String getCim() //Az ember címét adja vissza

virtual String getEmberTelefonszam() = 0 //Mindegyik származtatott osztálynak tudnunk kell majd a telefonszámát

size_t getMasAdatokElemszama() //Az ember egyéb adatainak elemszámát adja vissza

StringPar* getMasAdatokAdata(size_t index) //Az ember egyéb adatainak indexedik elemét adja vissza

void setNev(String _nev) //Az ember nevét módosítja

void setBecenev(String _becenev) //Az ember becenevét módosítja

void setCim(String _cim) //Az ember címét módosítja

virtual void setEmberTelefonszam(Telefonszam _telefonszam) = 0 //Módosíthatjuk a származtatott osztályok telefonszámát

void addMasAdatok(StringPar* sp) //Az ember egyéb adatai közül az indexedik elemét adja vissza

void setMasAdatok(size_t index, String adat) //Az ember egyéb adatai közül az indexedik elemét adja vissza

void removeMasAdatok(size_t index) //Az ember egyéb adatainak indexedik elemét eltávolítja

void kiir() //Az ember adatait egymás alá kiírja, minden sorban megjelenítjük, hogy az ember melyik adattagja jelenik meg abban a sorban

virtual String getTipus() = 0 //Megadja, hogy magánember, vagy dolgozó ember a példány

virtual ~Ember() //A masAdatok miatt kell

std::ostream& operator<<(std::ostream& os, Ember& ember) //Kiírja az ember adatait egymás mellé

Dolgozo : public Ember //Egy olyan ember adatait tárolja, akinek csak munkahelyi telefonszáma van

    adattagok:

        Telefonszam munkahelyiSzam //Egy ember céges telefonszámát

    metódusok:

        Dolgozo() //Létrehoz egy üres Dolgozo-t

        String getEmberTelefonszam() override //Az ember céges telefonszámát adja vissza

        void setEmberTelefonszam(Telefonszam _telefonszam) override //Az ember céges telefonszámát adja vissza

```



String getTipus() //Megadja, hogy a példány dolgozó

**Maganember** : public Ember //Egy olyan ember adatait tárolja, akinek csak munkahelyi telefonszáma van

adattagok:

Telefonszam privatSzam //Egy ember céges telefonszámát

metódusok:

Maganember() //Létrehoz egy üres Maganember-t

String getEmberTelefonszam() override //Az ember magán telefonszámát adja vissza

void setEmberTelefonszam(Telefonszam \_telefonszam) override //Az ember magán telefonszámát adja vissza

String getTipus() //Megadja, hogy a példány magánember

**Telefonkonyv** //Telefonkönyvek létrehozásához

adattagok:

Vektor<Ember\*> emberek //Ember tömbből áll a telefonkönyv, az emberek adatait tárolja, az Ember-ek felszabadítására az osztályon belül figyelünk

Vektor<String\*> egyebAdatok; //Egyéb adatok megnevezéseit tárolja

metódusok:

Telefonkonyv() //Létrehoz egy üres telefonkönyvet, nincs benne egy Ember sem

Ember\* getEmber(size\_t index) //Visszatér az indexedik ember objektum pointerével

Ember\* getEmber(const String& nev) //Visszatér a nev nevű objektum pointerével

size\_t getEmberekSzama() //Visszaadja, hogy hány ember van a telefonkönyvben

String\* getEgyebAdatok(size\_t index) //Visszaadja az indexedik egyéb adat nevét

size\_t getEgyebAdatokSzama() //Visszaadja, hogy hány ember van a telefonkönyvben

void addEmber(Ember\* ember) //A telefonkönyvhöz egy ember-t adunk hozzá

void addEgyebAdat(String\* nev) //A telefonkönyvhöz egy új adatot adunk hozzá

void removeEmber(String nev) //Törli a nev nevű embert a telefonkönyvből

void removeEgyebAdat(String\* nev) //Az egyéb adatok nev megnevezésű adatát eltávolítjuk

void kiir() //Kiírja az összes ember adatait, az első sorba az adattípusok szerepelnek

~Telefonkonyv(); //Az emberek és egyebAdatok miatt kell