

# Telefonkönyv

Lederer Márton Róbert projektje

## A program

A program egy telefonkönyv alkalmazás, mely alapvető digitális kezelést biztosít névjegyek és telefonszámok eltárolására.

A rendszer eltárolja a névjegyeket (fájlban is), amely tartalmazza a személy teljes nevét, becenevét, címét, valamint munkahelyi- és privát számát is. A névjegyek listázhatóak, lehet közöttük keresni, hozzá lehet adni újakat, valamint törölni is lehet őket.

## Használat

A program használata kizárólag a parancssoron keresztül működik, a felhasználó számára egyéb fájlok kezelése, létrehozása nem szükséges. Minden parancs a `telefonkonyv` főparancson keresztül érhető el. Alább láthatóak a támogatott parancsok, melyek minden inputot a standard inputról olvasnak be, ahol a részadatokat egyesével kéri be a rendszer.

## Új névjegy felvétele

A felhasználó legelső "feladata" egy új névjegy hozzáadása. Ez könnyen elvégezhető a következő paranccsal:

```
telefonkonyv add
```

A parancs futtatása után a program megkéri a felhasználót, hogy adja meg a névjegy adatait. A következő kimenet látható ekkor (itt példaként beírtam az adatokat):

```
Új névjegy felvétele:
```

```
Keresztnév: Marton
```

```
Vezetéknév: Lederer
```

Becenév: Marci  
Cím: 1037 Budapest, Szentendrei ut 57.  
Munkahelyi szám: +36 1 748 4519  
Privát szám: +36 90 123 9845

Az adatok megadása után az alkalmazás elmenti az új névjegyet.

## Névjegy törlése

A felhasználó bármikor törölhet egy névjegyet a telefonkönyvből:

```
telefonkonyv delete
```

A parancs beírása után a program felsorolja az eltárolt névjegyeket és megkéri a felhasználót, hogy adja meg, hányadik névjegyet szeretné törölni:

Névjegyek:

```
[1] Lederer Marton (+36 90 123 9845)
[2] Joska Pista (+36 70 834 1954)
[3] Kovacs Ferenc (+36 20 983 5533)
```

A törléshez kérem adja meg a törölni kívánt névjegy sorszámát: 1  
Névjegy törölve.

A kiválasztás után az alkalmazás eltávolítja a megadott névjegyet.

## Névjegyek listázása

A telefonkönyvben tárolt névjegyek listázhatóak az alábbi paranccsal:

```
telefonkonyv list
```

Ez a parancs kiírja az összes névjegyet, az összes hozzá tartozó adattal:

Névjegyek:

- [1] Lederer Marton (Marci)
  - Privát: +36 90 123 9845
  - Munkahelyi: +36 1 748 4519
  - Cím: 1037 Budapest, Szentendrei ut 57.
- [2] Joska Pista (Pisti)
  - Privát: +36 70 834 1954
  - Munkahelyi: +36 1 455 9158
  - Cím: 1035 Budapest, Harsfa utca 12.
- [3] Kovacs Ferenc (Feri)
  - Privát: +36 70 834 1954
  - Munkahelyi: +36 1 455 9158
  - Cím: 1035 Budapest, Harsfa utca 15.

## Névjegyek keresése

A névjegyek között lehet keresni a következő paranccsal:

```
telefonkonyv search
```

A parancs kér a felhasználótól egy kulcsszót, ami alapján keres a nevek és becenevek között:

Kulcsszó megadása: Marton

Keresés eredménye:

Lederer Marton (Marci)




- Privát: +36 90 123 9845
- Munkahelyi: +36 1 748 4519
- Cím: 1037 Budapest, Szentendrei ut 57.

## Telefonszám országoként statisztika

A felhasználó megtekintheti, hogy melyik országhoz hány telefonszám tartozik:

```
telefonkonyv stats
```

Statisztika:

-  Magyarország: 12
-  Egyesült Királyság: 3
-  Spanyolország: 1

## Algoritmusok

A program a `Loader` osztályon keresztül lesz vezérelhető, melynek funkcióit a `main()`-ben hívja majd meg, a felhasználó által megadott bemenetektől függően. Alább a program főbb algoritmusai olvashatók.

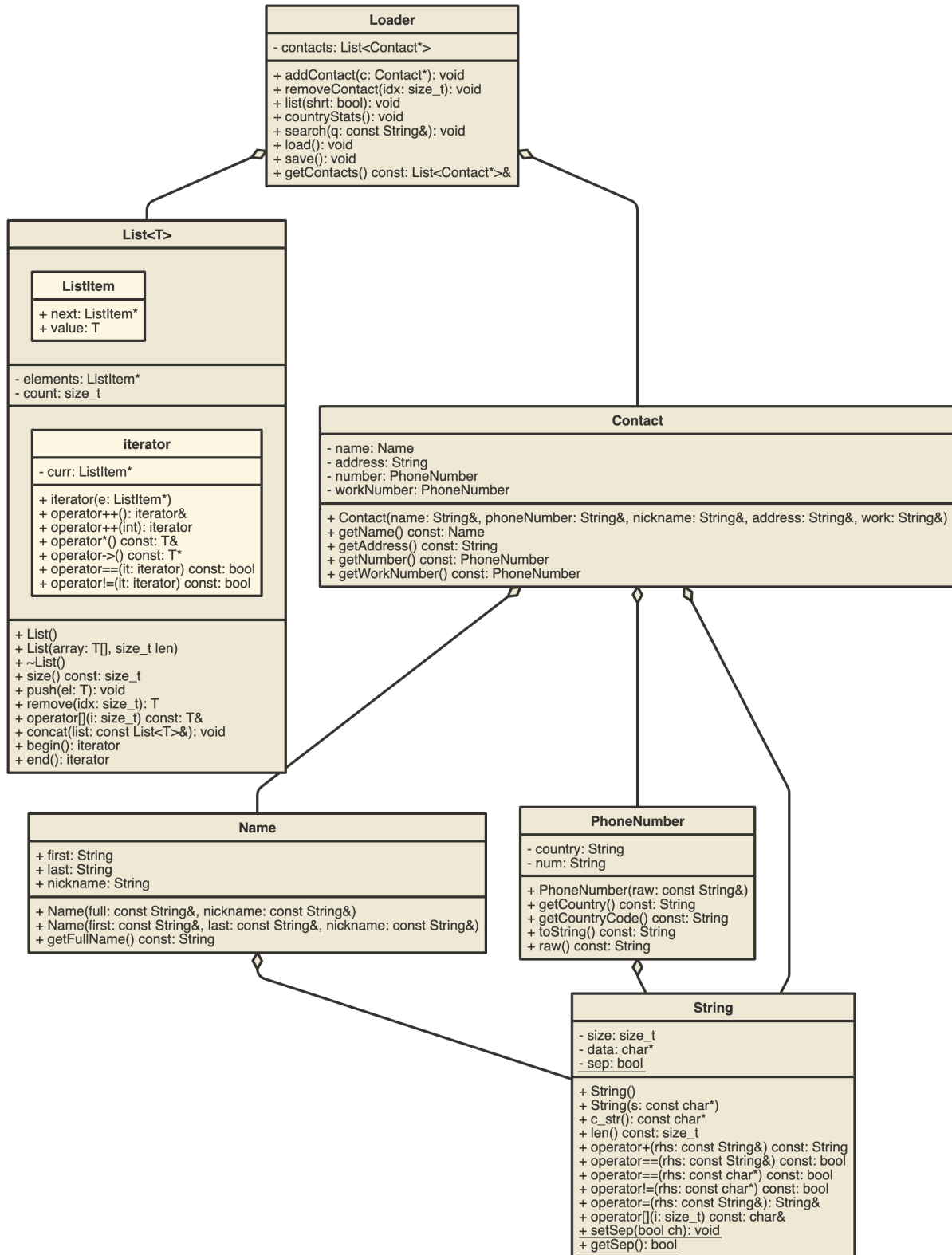
### Névjegykeresés algoritmus

A névjegykeresés úgy történik, hogy a program végigmegy az összes névjegyen és összehasonlítja a megadott kulcsszót a névjegyhez tartozó név stringekkel (csak kisbetűket használva). A neveknél a prioritás a következő: becenév > keresztnév > családnév. Az algoritmus implementációja a `Loader` osztályban lesz.

### Statisztika algoritmus

A program végigmegy a névjegyeken és összeszámolja, hogy melyik országhoz hány telefonszám tartozik.

## Osztálydiagram



# Megvalósítás

A megvalósításhoz szükséges volt egy saját `String`, illetve `List` (`Vector`) osztály elkészítése. Ezek az osztályok a C++ STL-ben való, bonyolultabb "testvéreik" működését mimikálja (a lista láncolt listás megoldást használva).

Az osztályok megvalósítása nagytöbbségében az előre definiált/meghatározott módon alakult. Változás csak néhány segédfunkció hozzáadásában, pár paraméter nevének megváltoztatásában, illetve pár típus átírásában volt.

A főbb változtatások:

- `List` destruktork hozzáadása
- `List concat()` funkció listák összefűzésére
- `List` típusdefiníciók az std kompatibilitáshoz
- `Loader list()` funkcióba opcionális `bool` paraméter, amivel minimalista stílusú telefonkönyv nyomtatható
- `Loader` destruktork hozzáadása
- `Name` egyparaméteres konstruktor eltávolítása (nem volt szükséges), referenciák
- `PhoneNumber raw()` funkció, nyers telefonszám `String` visszaadásához
- `String istream` sor szeparátor támogatása
- `String sub()` funkció substring készítéséhez

## Statisztika rangsorolása

A statisztika elkészítéséhez az eredeti terv szerint bináris beszűrásos rendezést használtam volna. A laboron való megbeszélés során úgy döntöttünk, hogy a statisztikához használható az STL is, így ez alapján az `std::sort` funkció használatával próbálkoztam.

Sajnos hamar kiderült számomra, hogy ehhez szükséges a random access iterátor támogatás, ugyanis a C++ STL `std::sort` gyorsrendezést használ, ami ezt az iterátor fajtát igényli.

Elkezdtem dolgozni a random access iterátor implementációján, azonban később az is kiderült, hogy ennek implementációja erősen korlátozott egy egyszerűen

láncolt listában, amin a `List` osztályom alapult.

Ez után megpróbálkoztam saját magam implementálni az algoritmust, de hasonló akadályokba ütköztem az egyszeres láncolás mellett. Sajnos a végső verzióban így nem tudtam megvalósítani a rangsorolást, bár a statisztika így is működik.

## Funkciók használata, osztálybemutató

A feladathoz 6 osztály készült el, valamint egy teszt program, ami akkor fut le, ha a CPORTA makró definiálva van. Egyébként a program normál konzoli applikáció üzemmódban fut, ahogyan az fentebb dokumentálva van. A teszteléshez a gtest lite könyvtárat használja, minden osztálynak leteszteli a fontosabb funkcióit.

Alább az osztályok bemutatása olvasható.

## String osztály

### Belső változók:

- `size_t size` - string mérete, '\0' nélkül
- `char* data` - a string C stringként
- `static bool sep` - szétválasztó karakter

### Konstruktor/destruktor:

- `String()` - Alapértelmezett konstruktor, üres string létrehozása.
- `String(const char* str)` - String készítése C stringből.
- `String(const char c)` - String készítése egyetlen karakterből.
- `String(const String& rhs)` - Másoló konstruktor, másik String példány másolása.
- `~String()` - Memória megszüntetését végzi.

### Tagfüggvények:

- `const char* c_str() const` - C string/char array készítése String példányból.
- `size_t len() const` - Méret visszaadása (lezáró nulla nélkül).
- `bool operator==(const String& rhs) const` - Egyenlőség vizsgálata két String között.
- `bool operator==(const char* rhs) const` - Egyenlőség vizsgálata String és C string között.

- `bool operator!=(const char* rhs) const` - Nem egyenlőség vizsgálata String és C string között.
- `String& operator=(const String& rhs)` - Értékadás operátor másik String példányból.
- `String& operator=(const char* rhs)` - Értékadás operátor C stringből.
- `String operator+(const String& rhs) const` - Két String összefűzése.
- `char& operator[](size_t idx) const` - Karakter elérése adott index alapján.
- `String sub(size_t idx, size_t len) const` - Substring készítése adott indextől, adott hosszúságban.
- `static void setSep(bool ch)` - Karakter beállítása, ahol olvasáskor véget ér a string (istream operátor használja).
- `static bool getSep()` - Szétválasztó karakter visszaadása.

### Globális függvények:

- `bool operator==(const char* lhs, const String& rhs)` - Egyenlőség vizsgálata C string és String között.
- `std::ostream& operator<<(std::ostream& os, const String& str)` - String kiírása az ostreamre.
- `std::istream& operator>>(std::istream& is, String& str)` - String beolvasása az istreamről.

## PhoneNumber osztály

### Belső változók:

- `String country` - Országkód tárolása.
- `String num` - Telefonszám tárolása.

### Konstruktor/destruktor:

- `PhoneNumber(const String& raw)` - Telefonszám feldolgozása string-ből.

### Tagfüggvények:

- `String raw() const` - Nem formázott telefonszám visszaadása.



- `String toString() const` - Telefonszám összeillesztése string-be.
- `String getCountry() const` - Ország nevének visszaadása.
- `String getCountryCode() const` - Országkód visszaadása.

## Name osztály

### Belső változók:

- `String first` - Keresztnév tároló.
- `String last` - Vezetéknév tároló.
- `String nickname` - Becenév tároló.

### Konstruktor/destruktor:

- `Name(const String& full, const String& nickname)` - Teljes név és becenév feldolgozása.
- `Name(const String& first, const String& last, const String& nickname)` - Keresztnév, vezetéknév és becenév beállítása.

### Tagfüggvények:

- `String getFullName() const` - Teljes név összeillesztése.

## Loader osztály

### Belső változók:

- `List<Contact*> contacts` - Névjegyek tárolása, a telefonkönyv tartalma.

### Konstruktor/destruktor:

- `~Loader()` - Memória megszüntetését végzi.

### Tagfüggvények:

- `void addContact(Contact* c)` - Új névjegy felvétele.
- `void removeContact(size_t idx)` - Névjegy eltávolítása adott index alapján.
- `void list(bool shrt = false)` - Névjegyek listázása, rövid vagy részletes formában (a rövid formát csak a névjegytörlésnél használja a program).

- `void countryStats()` - Statisztika készítése, hogy melyik országhoz hány telefonszám tartozik.
- `void search(const String& q)` - Névjegy keresése szöveg alapján.
- `void load()` - Névjegyek betöltése fájlból.
- `void save()` - Névjegyek mentése fájlba.
- `List<Contact*>& getContacts()` - Névjegyek visszaadása.

## List osztály

### Belső változók:

- `ListItem* elements` - A láncolt lista elemeinek kezdőpontja.
- `size_t count` - Lista aktuális mérete.

### Konstruktor/destruktor:

- `List()` - Alapértelmezett konstruktor, üres lista létrehozása.
- `List(T array[], size_t len)` - Lista létrehozása tömbből a megadott mérettel.
- `~List()` - Memória megszüntetését végzi.

### Tagfüggvények:

- `size_t size() const` - Lista méretének visszaadása.
- `void push(T el)` - Új elem hozzáadása a listához.
- `T remove(size_t idx)` - Elem törlése a listáról index szerint.
- `T& operator[](size_t idx) const` - Elem visszaadása index szerint.
- `iterator begin()` - Iterátor kezdőpont inicializálása.
- `iterator end()` - Iterátor végpont inicializálása.
- `void concat(List<T>& list)` - Egy másik lista hozzáfűzése ehhez a listához.

### Iterátor:

- `iterator(ListItem* e = nullptr)` - Iterátor inicializálása.
- `iterator& operator++()` - Iterátor előreléptetése.
- `iterator operator++(int)` - Iterátor előreléptetése utánnöveléssel operátorral.

- `T& operator*() const` - Elem értékéhez való hozzáférés.
- `T* operator->() const` - Elem mutatójának elérése.
- `bool operator==(iterator it) const` - Iterátorok egyenlőségének vizsgálata.
- `bool operator!=(iterator it) const` - Iterátorok nem egyenlőségének vizsgálata.

## Contact osztály

### Belső változók:

- `Name name` - Név tárolása.
- `String address` - Cím tárolása.
- `PhoneNumber number` - Mobiltelefonszám.
- `PhoneNumber workNumber` - Munkahelyi telefonszám.

### Konstruktor/destruktor:

- `Contact(String& name, String& phoneNumber, String& nickname, String& address, String& work)` - Névjegy létrehozása névvel, mobilszámmal, becenévvel, címmel és munkahelyi számmal.

### Tagfüggvények:

- `Name getName() const` - Név visszaadása.
- `String getAddress() const` - Cím visszaadása.
- `PhoneNumber getNumber() const` - Mobilszám visszaadása.
- `PhoneNumber getWorkNumber() const` - Munkahelyi telefonszám visszaadása.

### Globális függvények:

- `std::ostream& operator<<(std::ostream& os, const Contact& c)` - Kontakt adatainak kiírása az ostreamre.