HÁZI FELADAT

Programozás alapjai 2.

Dokumentáció

Benedek Olivér
C2JUZ6

2023. április 15.

TARTALOM

1. Feladat 2
2. Feladatspecifikáció 2
3. Pontosított feladatspecifikáció 3
4. Terv 3
5. Tesztprogram bemutatása 5
6. A program interfésze: 6
7. Mellékletek 6

1. Feladat

Telefonkönyv

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!

2. Feladatspecifikáció

A telefonkönyv program parancssoros és menüvezérelt. Az egyes menüpontokat a megfelelő billentyű leütésével tudja elérni a felhasználó.
A menüpontok a következők:

- Összes adat listázása
- Keresés
- Adatrögzítés
- Törlés
- Mentés
- Kilépés

A program a szabványos bemenetről olvas be adatokat, ezeket dinamikusan tárolja (nincs korlát a méretükben). Az adatok helyességéért a felhasználó a felelős.
A programban nincs automatikus mentés, ezt a felhasználó tudja manuálisan megtenni, azonban kilépéskor a program megkérdezi, hogy tervez-e menteni.

Az adatok módosítására lehetőség nincs, csak törölni lehet azokat, majd újból rögzíteni.

Tesztelésre egy olyan programot használok, mely először valódi, majd hibás adatokkal próbálja végig a funkciókat, ezzel szemléltetve a kivételkezelést.

3. Pontosított feladatspecifikáció

A telefonkönyv program parancssoros és menüvezérelt. Az egyes menüpontokat a megfelelő billentyű leütésével tudja elérni a felhasználó.
A menüpontok a következők:

- Összes adat listázása
- Keresés
- Adatrögzítés
- Törlés
- Adatok módosítása
- Mentés
- Kilépés

A program a szabványos bemenetről olvas be adatokat, ezeket dinamikusan tárolja (nincs korlát a méretükben). Az adatok helyességéért a felhasználó a felelős.
A programban nincs automatikus mentés, ezt a felhasználó tudja manuálisan megtenni, azonban kilépéskor a program megkérdezi, hogy tervez-e menteni.

Tesztelésre egy olyan programot használok, mely először valódi, majd hibás adatokkal próbálja végig a funkciókat, ezzel szemléltetve a kivételkezelést.

4. Terv

A feladathoz egy osztálydiagramot készítettem, mely megmutatja az egyes osztályok hierarchiáját, valamint azok függvényeit.
A telefonkönyv a *Konyv* osztály dinamikus tömbjéből fog felépülni. Ebben a tömbben *Szemely* elemek vannak, mely az egyes emberek adatait tárolják. Az adatok három fajta osztályból épülnek fel: *Cim*, *Szam*, *Nev*. Ezek az osztályok az 5. laboron elkészített *String* osztály használatával kezelik az adatokat, ezzel is színesítve az osztályokat.
A *Cim* osztályba kerülnek bele a következő adatok: irányítószám, ország, város, utcanév és házszám. A *Szam* osztály privát és munkahelyi számokat tárol, míg a *Nev* vezeték -és keresztnevet foglal magába.

Több tesztprogramot fogok elkészíteni, melyek végigmennek az egyes menüpontokon, és minden függvényt meghívnak.

A program fő függvényei:

listazas() – végigmegy a tömbön a 0. elemtől az utoljára rögzített elemig, és minden elem adatait a szabványos kimenetre írja.

kereses() – bekéri a felhasználótól, hogy mi alapján szeretne keresni, majd a találat eredményét a szabványos kimenetre írja. Amennyiben nincs találat, azt jelzi a felhasználónak.

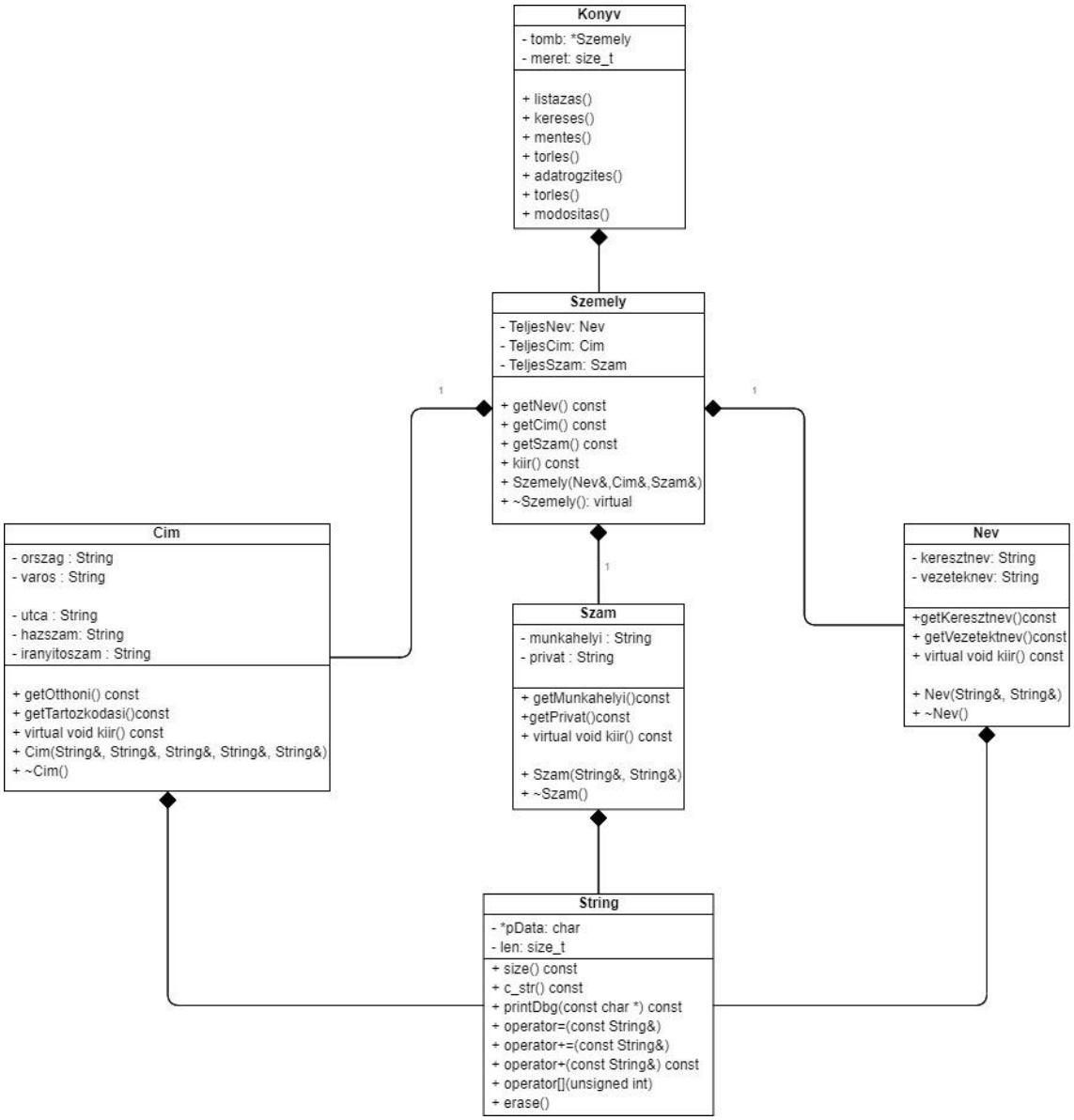
adatrogzites() – egy *Szemely* objektumot a dinamikus tömb végére rak, ha a tömbben nincsen hely, akkor újra foglal egy nagyobb területet, és átmásolja oda a tömböt.

mentes() – A tömb összes adatát fájlba írja.

torles() – bekéri a felhasználótól a törölni kívánt *Szemely* objektum adatait, majd kitörli azt a tömbből, és a többi adatot is úgy rendezi, hogy továbbra is használható maradjon a tömb.

modositas() – bekéri a módosítani kívánt objektumot, és annak módosításait, majd megkeresi azt a tömbben, és felülírja az új adatokkal.

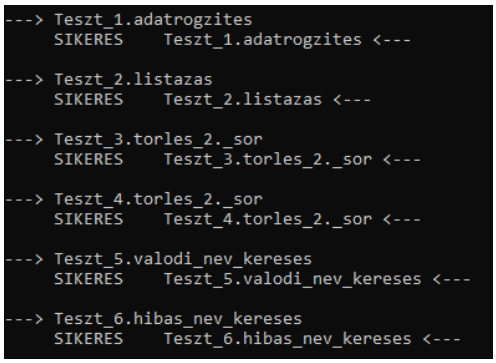
Osztálydiagram:



5. Tesztprogram bemutatása

A tesztelést két fázisban készítettem el:

1. A laborokon is használt *gtest_lite.h* program segítségével tudtam ellenőrizni azokat a függvényeket, amelyeknek van visszatérési értéküket, és azokat össze lehet vetni más adatokkal a teszteléshez.
   Ezt a tesztelést a *teszt_1* függvény segítségével hajtom végre, amiben 6 teszt van.



   Teszt 1: Adatrögzítés tesztje:

   - Ellenőrzi, hogy a *telefonkonyv* méretét.
   - Hozzáad egy új személyt a *telefonkonyv* objektumhoz.
   - Ellenőrzi, hogy a *telefonkonyv* mérete növekedett.

   Teszt 2: Listázás tesztje:

   - Listázza a telefonkonyv tartalmát, majd összehasonlítja a kapott stringet a várt stringgel.

   Teszt 3-4: Törlés tesztje:

   - Törli a telefonkonyv második sorát.
   - Ellenőrzi, hogy a telefonkonyv mérete csökkent.
   - Listázza a telefonkonyv tartalmát, majd összehasonlítja a kapott stringet a várt stringgel.

   Teszt 5: Valódi név keresés tesztje:

   - Keresi a telefonkonyv a megadott névvel rendelkező személyt.
   - Listázza a talált sorokat, és összehasonlítja a kapott stringet a várt stringgel.

   Teszt 6: Hibás név keresés tesztje:

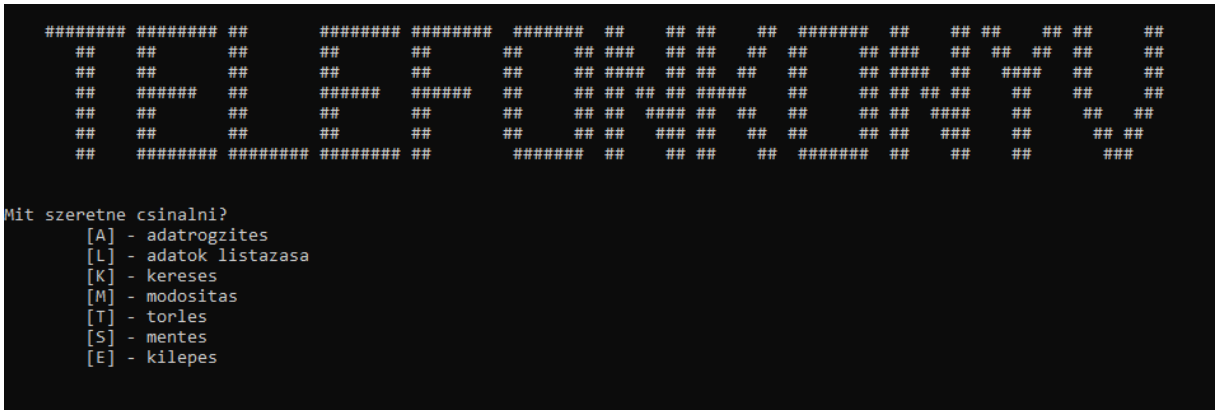
   - Keresi a telefonkonyv a nem létező névvel rendelkező személyt.
   - Listázza, hogy nincs találat.

2. A *szabvanyos_bemenet.txt* segítségével végigmegy a menü összes pontján, és leteszteli annak minden funkcióját, amit a *teszt_1* függvény eddig nem tesztelt volna. Ezek között van a módosítás, keresés telefonszám szerint és törlés.

Memóriakezelés tesztje

A memóriakezelés ellenőrzését a laborgyakorlatokon használt MEMTRACE modullal végeztem. A modul működésének érdekében a memtrace.h fájlt minden fájlban include-oltam. A modul nem jelzett memóriaszivárgást.

6. A program interfésze:



7. Mellékletek

konyv.cpp

```
#include <iostream>
#include <sstream>
#include <fstream>

#include "konyv.h"
#include "memtrace.h"

using std::cin;
using std::cout;

void Konyv::adatrogzites(const Szemely& szemely) {
    // Új tömb létrehozása a meglévő méret + 1 elemmel
    Szemely* ujTomb = new Szemely[meret + 1];
    // Az aktuális elemek másolása az új tömbbe
    for (size_t i = 0; i < meret; i++) {
        ujTomb[i] = tomb[i];
    }
    // Az új elem hozzáadása az új tömbhöz
    ujTomb[meret] = szemely;
    // A régi tömb felszabadítása
    delete[] tomb;
    // Az új tömb beállítása a 'tomb' mutatóra
    tomb = ujTomb;
    // A tömb méretének növelése
    meret++;
}

void Konyv::adatrogzites() {
    // Új személy objektum létrehozása és beolvasása
    Szemely uj;
    uj = uj.szemely_beolvas();
    // Új tömb létrehozása a meglévő méret + 1 elemmel
    Szemely* ujTomb = new Szemely[meret + 1];
    // Az aktuális elemek másolása az új tömbbe
    for (size_t i = 0; i < meret; i++) {
        ujTomb[i] = tomb[i];
    }
    // Az új elem hozzáadása az új tömbhöz
    ujTomb[meret] = uj;
    // A régi tömb felszabadítása
    delete[] tomb;
    // Az új tömb beállítása a 'tomb' mutatóra
    tomb = ujTomb;
    // A tömb méretének növelése
    meret++;
}

std::ostream& Konyv::listazas(std::ostream& os){
    // Új sor a kimeneten
    os<<"\n";
    // Minden elem kiírása a tömbből
    for(size_t i = 0; i<meret;i++)
```

```
        tomb[i].szemely_kiir(os);
    // Új sor a kimeneten
    os<<"\n";
    return os;
}

std::ostream& Konyv::kereses(std::ostream& os){
    // Felhasználói bemenet fogadása a keresés típusához
    cout<<"\n\t[P] - Privat szam\n\t[M] - Munkahelyi szam\n\t[N] - Nev\n";
    char bemenet;
    cin>>bemenet;
    bemenet= tolower(bemenet);
    bool talalat = false;
    String p;
    String m;
    Nev n;
    switch(bemenet){
    case 'p':
        cout<<"Privat szam: ";
        cin>>p;
        // A találatok kiírása, ha megtalálható a privat szám a tömbben
        for(size_t i = 0; i<meret;i++)
            if(tomb[i].getTeljesSzam().getPrivat()==p){
                tomb[i].szemely_kiir(os);
                talalat =true;
            }
        break;
    case 'm':
        cout<<"Munkahelyi szam: ";
        cin>>m;
        // A találatok kiírása, ha megtalálható a munkahelyi szám a tömbben
        for(size_t i = 0; i<meret;i++)
            if(tomb[i].getTeljesSzam().getMunkahelyi()==m){
                tomb[i].szemely_kiir(os);
                talalat =true;
            }
        break;

    case 'n':
        n=n.nev_beolvas();
        // A találatok kiírása, ha megtalálható a név a tömbben
        for(size_t i = 0; i<meret;i++)
            if(tomb[i].getTeljesNev().getKeresztnev()==n.getKeresztnev() &&
tomb[i].getTeljesNev().getVezetektnev()==n.getVezetektnev()){
                tomb[i].szemely_kiir(os);
                talalat =true;
            }
        break;
    }
    // Ha nincs találat, kiírás
    if(!talalat)
        os<<"\nNincs talalat.\n";
    return os;
}

std::ostream& Konyv::kereses_nev(std::ostream& os,Nev& n){
    bool talalat = false;
    // A találatok kiírása, ha megtalálható a név a tömbben
    for(size_t i = 0; i<meret;i++)
        if(tomb[i].getTeljesNev().getKeresztnev()==n.getKeresztnev() &&
tomb[i].getTeljesNev().getVezetektnev()==n.getVezetektnev()){
            // Az első találatnál kiírás, hogy talált sorok vannak
            if(!talalat)
                os<<"\nTalalt sor(ok):\n";
            tomb[i].szemely_kiir(os);
            talalat =true;
        }
    // Ha nincs találat, kiírás
    if(!talalat)
        os<<"\nNincs talalat.\n";
    return os;
}

size_t Konyv::elso_talalat(){
    // Felhasználói bemenet fogadása a keresés típusához
    cout<<"\n\t[P] - Privat szam\n\t[M] - Munkahelyi szam\n\t[N] - Nev\n";
    char bemenet; cin>>bemenet;
    bemenet = tolower(bemenet);
    String p;
    String m;
    Nev n;
    switch(bemenet){
    case 'p':
        cout<<"Privat szam: ";
        cin>>p;
        // Az első találat sorszáma visszaadása, ha megtalálható a privat szám a tömbben
        for(size_t i = 0; i<meret;i++)
            if(tomb[i].getTeljesSzam().getPrivat()==p)
                return i;
        break;
    case 'm':
```

```
        cout<<"Munkahelyi szam: ";
        cin>>m;
        // Az első találat sorszáma visszaadása, ha megtalálható a munkahelyi szám a tömbben
        for(size_t i = 0; i<meret;i++)
            if(tomb[i].getTeljesSzam().getMunkahelyi()==m )
                return i;
        break;
    case 'n':
        n=n.nev_beolvas();
        // Az első találat sorszáma visszaadása, ha megtalálható a név a tömbben
        for(size_t i = 0; i<meret;i++)
            if(tomb[i].getTeljesNev().getKeresztnev()==n.getKeresztnev())
                return i;
        break;
    }
    // Ha nincs találat, -1 visszaadása
    return -1;
}

std::ostream& Konyv::torles(std::ostream& os,size_t i){
    // Ha nincs találat, kiírás és visszatérés
    if(i==static_cast<size_t>(-1)){
        os<<"\nNincs talalat.\n";
        return os;
    }
    // A törlendő sor kiírása
    os<<"Torlesre kerulo sor: ";
    tomb[i].szemely_kiir(os);
    // Méret csökkentése
    meret--;
    // A tömb rendezése a törlendő sor utáni sorok törlése után
    for(size_t j = i; j < meret; j++)
        tomb[j] = tomb[j + 1];
    return os;
}

void Konyv::modositas(size_t i){
    // Ha nincs találat, kiírás és visszatérés
    if(i==static_cast<size_t>(-1)){
        cout<<"\nNincs talalat.\n";
        return;
    }
    // A módosítandó sor kiírása
    cout<<"Modositando sor: ";
    tomb[i].szemely_kiir(cout);
    cout<<"\nMit szeretne modositani rajta?\n1 - keresztnev\n2 - vezeteknev\n3 - privat szam\n4 - munkahelyi
szam\n5 - orszag\n6 - varos\n7 - utca\n8 - hazszam\n9 - iranyitoszam\n";
    int bemenet; cin>>bemenet;
    cout<<"Uj ertek: ";
    String ujadat; cin>>ujadat;
    switch(bemenet){
    case 1:
        tomb[i].getteljesnev().setKeresztnev(ujadat);
        break;
    case 2:
        tomb[i].getteljesnev().setVezeteknev(ujadat);
        break;
    case 3:
        tomb[i].getteljesszam().setPrivat(ujadat);
        break;
    case 4:
        tomb[i].getTeljesSzam().setMunkahelyi(ujadat);
        break;
    case 5:
        tomb[i].getteljescim().setOrszag(ujadat);
        break;
    case 6:
        tomb[i].getteljescim().setVaros(ujadat);
        break;
    case 7:
        tomb[i].getteljescim().setUtca(ujadat);
        break;
    case 8:
        tomb[i].getteljescim().setHazszam(ujadat);
        break;
    case 9:
        tomb[i].getteljescim().setIranyitoszam(ujadat);
        break;
    }
}

/**
 * A `mentes` függvény felelős a telefonkönyv tartalmának mentéséért.
 * Az adatokat egy "telefonkonyv.txt" nevű fájlba írja ki tabulátorral elválasztva.
 * A függvény végigmegy a `tomb` tömbön és minden személy adatait kiírja a fájlba.
 */
void Konyv::mentes() {
    std::ofstream file("telefonkonyv.txt");
    if (file.is_open()) {
        for (size_t i = 0; i < meret; ++i) {
            // A személy adatait tabulátorral elválasztva írjuk ki a fájlba
```

```
            file << tomb[i].getTeljesNev().getVezetektnev() << "\t"
                 << tomb[i].getTeljesNev().getKeresztnev() << "\t"
                 << tomb[i].getTeljesSzam().getPrivat() << "\t"
                 << tomb[i].getTeljesSzam().getMunkahelyi() << "\t"
                 << tomb[i].getTeljesCim().getOrszag() << "\t"
                 << tomb[i].getTeljesCim().getVaros() << "\t"
                 << tomb[i].getTeljesCim().getUtca() << "\t"
                 << tomb[i].getTeljesCim().getHazszam() << "\t"
                 << tomb[i].getTeljesCim().getIranyitoszam() << "\t"
                 << std::endl;
        }
        file.close();
    }
}

/**
 * A `betoltes` függvény felelős a telefonkönyv tartalmának betöltéséért.
 * A függvény beolvassa az adatokat a "telefonkonyv.txt" nevű fájlból és feltölti velük a `tomb` tömböt.
 */
void Konyv::betoltes() {
    std::ifstream file("telefonkonyv.txt");
    if (file.is_open()) {
        std::string line;
        size_t i = 0;
        while (std::getline(file, line)) {
            std::istringstream iss(line);
            std::string vezeteknev, keresztnev, privat, munkahelyi, orszag, varos, utca, hazszam,
iranyitoszam;
            // Az adatokat tabulátorral elválasztva beolvassuk a megfelelő változókba
            if (iss >> vezeteknev >> keresztnev >> privat >> munkahelyi >> orszag >> varos >> utca >>
hazszam >> iranyitoszam) {
                // Az adatokat felhasználva létrehozzuk egy Szemely objektumot és beillesztjük a `tomb`
tömbbe
                tomb[i] = Szemely(Nev(vezeteknev, keresztnev), Szam(privat, munkahelyi), Cim(orszag, varos,
utca, hazszam, iranyitoszam));
                i++;
            }
        }
        file.close();
    }
}

/**
 * A `sor_szamol` függvény felelős a sorok számának meghatározásáért a "telefonkonyv.txt" fájlban.
 * Beolvassa a fájlt soronként és minden sor beolvasása után növeli a `darab` változót.
 * Végül visszatér a darab értékével, ami az összes sor számát jelenti.
 */
size_t sor_szamol() {
    size_t darab = 0;
    std::string line;
    std::ifstream file("telefonkonyv.txt");
    if (file.is_open()) {
        while (std::getline(file, line)) {
            // Minden sor beolvasása után növeljük a darab változót
            ++darab;
        }
        file.close();
    }
    return darab;
}
```

main.cpp

```
#include <iostream>
#include "konyv.h"

using std::cout;
using std::cin;

#include "memtrace.h"
#include "gtest_lite.h"


void teszt_1(Konyv& telefonkonyv)
{
    TEST(Teszt_1, adatrogzites)
    EXPECT_EQ(1, telefonkonyv.getMeret()) << "Rossz meret" << std::endl;

telefonkonyv.adatrogzites(Szemely(Nev("Teszt","Istvan"),Szam("061234569","+367210124"),Cim("Magyarorszag","S
zeged","Petofi utca","98","4391")));
    EXPECT_EQ(2, telefonkonyv.getMeret()) << "Rossz meret" << std::endl;
    Szemely sz =
Szemely(Nev("Pelda","Jozsef"),Szam("+362398741","062381301"),Cim("Magyarorszag","Szekesfehervar","Jozsef
Attila utca","1","1234"));
    telefonkonyv.adatrogzites(sz);
    EXPECT_EQ(3, telefonkonyv.getMeret()) << "Rossz meret" << std::endl;
    END

    TEST(Teszt_2, listazas)
```

```
    std::stringstream ss;
    telefonkonyv.listazas(ss);
    std::string s = ss.str();
    std::string str = "\nBenedek Oliver\t0612345678 +3630124234\tMagyarorszag 17 Budapest Irinyi utca\nTeszt
Istvan\t061234569 +367210124\tMagyarorszag 4391 Szeged Petofi utca 98\nPelda Jozsef\t+362398741
062381301\tMagyarorszag 1234 Szekesfehervar Jozsef Attila utca 1\n\n";
    EXPECT_EQ(str,s) << "Hibas listazas" << std::endl;
    END

    TEST(Teszt_3, torles_2._sor)
    std::stringstream dump;
    telefonkonyv.torles(dump,1);
    EXPECT_EQ(2, telefonkonyv.getMeret()) << "Rossz meret" << std::endl;
    std::stringstream ss;
    telefonkonyv.listazas(ss);
    std::string s = ss.str();
    std::string str = "\nBenedek Oliver\t0612345678 +3630124234\tMagyarorszag 17 Budapest Irinyi utca\nPelda
Jozsef\t+362398741 062381301\tMagyarorszag 1234 Szekesfehervar Jozsef Attila utca 1\n\n";
    EXPECT_EQ(str,s) << "Hibas listazas" << std::endl;
    END

    TEST(Teszt_4, torles_2._sor)
    std::stringstream dump;
    telefonkonyv.torles(dump,1);
    EXPECT_EQ(1, telefonkonyv.getMeret()) << "Rossz meret" << std::endl;
    std::stringstream ss;
    telefonkonyv.listazas(ss);
    std::string s = ss.str();
    std::string str = "\nBenedek Oliver\t0612345678 +3630124234\tMagyarorszag 17 Budapest Irinyi utca\n\n";
    EXPECT_EQ(str,s) << "Hibas listazas" << std::endl;
    END

    TEST(Teszt_5, valodi_nev_kereses)
    Nev valodinev = Nev("Benedek","Oliver");
    std::stringstream ss;
    telefonkonyv.kereses_nev(ss,valodinev);
    std::string s = ss.str();
    std::string str = "\nTalalt sor(ok):\nBenedek Oliver\t0612345678 +3630124234\tMagyarorszag 17 Budapest
Irinyi utca\n";
    EXPECT_EQ(str,s) << "Hibas kereses" << std::endl;
    END

    TEST(Teszt_6, hibas_nev_kereses)
    Nev hibasnev = Nev("Nincs","Ilyen");
    std::stringstream ss;
    telefonkonyv.kereses_nev(ss,hibasnev);
    std::string s = ss.str();
    std::string str = "\nNincs talalat.\n";
    EXPECT_EQ(str,s) << "Hibas kereses" << std::endl;
    END
    cout<<"\n---> Teszt_6.Mentes\n";
    telefonkonyv.mentes();
}

int main()
{
    // Sorok számának meghatározása
    size_t sorok = sor_szamol();

    // Konyv objektum létrehozása a meghatározott sorok számával
    Konyv telefonkonyv(sorok);

    // Telefonkönyv betöltése
    telefonkonyv.betoltes();

    //Tesztelés
    teszt_1(telefonkonyv);

bool futas = true;
while (futas == true) {
    char betu;

    // Felhasználótól bekérjük a választ a kívánt mûvelethez
    cout << "\nMit szeretne csinalni?\n\t[A] - adatrogzites\n\t[L] - adatok listazasa\n\t[K] -
kereses\n\t[M] - modositas\n\t[T] - torles\n\t[S] - mentes\n\t[E] - kilepes\n";
    cin >> betu;
    betu = tolower(betu);

    // A bekért választ alapján elágazunk a megfelelõ mûvelet végrehajtásához
    switch (betu) {
        case 'a':
            // Adat rögzítése
            telefonkonyv.adatrogzites();
            break;
        case 'l':
            // Adatok listázása
            telefonkonyv.listazas(std::cout);
            break;
        case 'k':
            // Keresés
            cout << "\nMi alapjan szeretne keresni?";
```

```
            telefonkonyv.kereses(std::cout);
            break;
        case 't':
            // Törlés
            cout << "\nMi alapjan szeretne torolni?";
            telefonkonyv.torles(cout, telefonkonyv.elso_talalat());
            break;
        case 'm':
            // Módosítás
            cout << "\nMi alapjan szeretne modositani?";
            telefonkonyv.modositas(telefonkonyv.elso_talalat());
            break;
        case 's':
            // Mentés
            telefonkonyv.mentes();
            break;
        case 'e':
            // Kilépés
            futas = false;
            break;
        }
    }
    return 0;
}
```

nev.cpp

```
#include "nev.h"
#include "memtrace.h"

/**
 * Az `operator<<` függvény a `Nev` objektumot írja ki a megadott kimeneti adatfolyamra.
 * Az objektum `vezetektnev` és `keresztnev` adattagjait írja ki szóközzel elválasztva.
 * A függvény visszatérési értéke a kimeneti adatfolyam referenciája, amely lehetõvé teszi a láncolt írást.
 *
 * @param os A kimeneti adatfolyam, amelyre az objektum kiírásra kerül.
 * @param n A `Nev` objektum, amelyet ki szeretnénk írni.
 * @return A kimeneti adatfolyam referenciája.
 */
std::ostream& operator<<(std::ostream& os,const Nev& n){
    os<<n.getVezetektnev()<<" "<<n.getKeresztnev()<<"\t";
    return os;
}
```

cim.cpp

```
#include "cim.h"
#include "memtrace.h"

/**
 * Az `operator<<` függvény a `Cim` objektumot írja ki a megadott kimeneti adatfolyamra.
 * Az objektum `orszag`, `iranyitoszam`, `varos`, `utca` és `hazszam` adattagjait írja ki szóközzel
elválasztva.
 * A függvény visszatérési értéke a kimeneti adatfolyam referenciája, amely lehetõvé teszi a láncolt írást.
 *
 * @param os A kimeneti adatfolyam, amelyre az objektum kiírásra kerül.
 * @param c A `Cim` objektum, amelyet ki szeretnénk írni.
 * @return A kimeneti adatfolyam referenciája.
 */
std::ostream& operator<<(std::ostream& os,const Cim& c){
    os<<c.getOrszag()<<" "<<c.getIranyitoszam()<<" "<<c.getVaros()<<" "<<c.getUtca()<<" "<<c.getHazszam();
    return os;
}
```

szam.cpp

```
#include "szam.h"
#include "memtrace.h"

/**
 * Az `operator<<` függvény a `Szam` objektumot írja ki a megadott kimeneti adatfolyamra.
 * Az objektum `privat` és `munkahelyi` adattagjait írja ki szóközzel elválasztva.
 * A függvény visszatérési értéke a kimeneti adatfolyam referenciája, amely lehetõvé teszi a láncolt írást.
 *
 * @param os A kimeneti adatfolyam, amelyre az objektum kiírásra kerül.
 * @param s A `Szam` objektum, amelyet ki szeretnénk írni.
 * @return A kimeneti adatfolyam referenciája.
 */
std::ostream& operator<<(std::ostream& os,const Szam& s){
    os<<s.getPrivat()<<" "<<s.getMunkahelyi()<<"\t";
    return os;
}
```

konyv.h

```
#ifndef KONYV_H
#define KONYV_H

#include "szemely.h"

/**
 * A `Konyv` osztály egy telefonkönyvet reprezentál.
 * Tartalmazza a könyv méretét (`meret`) és a személyek tömbjét (`tomb`).
 */
class Konyv {
    size_t meret;       // A telefonkönyv mérete
    Szemely* tomb;      // A személyek tömbje

public:
    /**
     * Az `Konyv` osztály konstruktora.
     * Az osztály példányosításakor inicializálja a méretet és létrehozza a személyek tömbjét.
     */
    Konyv(size_t m = 0) : meret(m), tomb(new Szemely[m]) {}

    // Telefonkönyv betöltése a fájlból
    void betoltes();
    // Telefonkönyv listázása
    std::ostream& listazas(std::ostream&);
    // Új személy hozzáadása a telefonkönyvhöz
    void adatrogzites(const Szemely&);
    // Felhasználótól adatok bekérése és hozzáadása a telefonkönyvhöz
    void adatrogzites();
    // Személyek keresése a telefonkönyvben
    std::ostream& kereses(std::ostream&);
    // Személyek keresése név alapján a telefonkönyvben
    std::ostream& kereses_nev(std::ostream&, Nev&);
    // Az elsõ találat indexét adja vissza
    size_t elso_talalat();
    // Telefonkönyv mentése fájlba
    void mentes();
    // Személy törlése a telefonkönyvbõl
    std::ostream& torles(std::ostream&, size_t);
    // Személy adatainak módosítása
    void modositas(size_t);

    size_t getMeret() const { return meret; }
    // A méret lekérdezése

    // Az [] operátor túlterhelése, hogy az osztály példányát használva lehessen az elemekhez hozzáférni
    Szemely operator[](size_t i) const { return tomb[i]; }
    Szemely& operator[](size_t i) { return tomb[i]; }

    /**
     * Az `Konyv` osztály destruktora.
     * Felszabadítja a dinamikusan lefoglalt tömb memóriaterületét.
     */
    ~Konyv() { delete[] tomb; }
};

/**
 * A `sor_szamol` függvény felelõs a sorok számának meghatározásáért a "telefonkonyv.txt" fájlban.
 * Beolvassa a fájlt soronként és minden sor beolvasása után növeli a `darab` változót.
 * Végül visszatér a darab értékével, ami az összes sor számát jelenti.
 */
size_t sor_szamol();


#endif
```

szemely.h

```
#ifndef SZEMELY_H
#define SZEMELY_H

#include "nev.h"
#include "szam.h"
#include "cim.h"

class Szemely {
    Nev teljesNev;  // Személy teljes neve
    Szam teljesSzam;  // Személy telefonszámai
    Cim teljesCim;  // Személy lakcíme
public:
    /// Alapértelmezett konstruktor
```

```
    Szemely() {}

    /// Konstruktor, amely beállítja a személy teljes nevét, telefonszámait és lakcímét
    Szemely(Nev n, Szam sz, Cim c) : teljesNev(n), teljesSzam(sz), teljesCim(c) {}

    /// Értékadás operátor
    Szemely& operator=(const Szemely& sz) {
        if (this != &sz) {
            teljesNev = sz.teljesNev;
            teljesSzam = sz.teljesSzam;
            teljesCim = sz.teljesCim;
        }
        return *this;
    }

    /// Getter függvény, visszaadja a személy teljes nevét
    Nev getTeljesNev() const { return teljesNev; }

    /// Getter függvény, visszaadja a személy teljes nevét referenciaként
    Nev& getteljesnev() { return teljesNev; }

    /// Getter függvény, visszaadja a személy telefonszámait
    Szam getTeljesSzam() const { return teljesSzam; }

    /// Getter függvény, visszaadja a személy telefonszámait referenciaként
    Szam& getteljesszam() { return teljesSzam; }

    /// Getter függvény, visszaadja a személy lakcímét
    Cim getTeljesCim() const { return teljesCim; }

    /// Getter függvény, visszaadja a személy lakcímét referenciaként
    Cim& getteljescim() { return teljesCim; }

    /// Beolvas egy személyt a felhasználótól
    Szemely szemely_beolvas() {
        Nev n;
        Szam sz;
        Cim c;
        n = n.nev_beolvas();
        sz = sz.szam_beolvas();
        c = c.cim_beolvas();
        return Szemely(n, sz, c);
    }

    /// Kiírja a személyt a megadott stream-re
    std::ostream& szemely_kiir(std::ostream& os) {
        return os << teljesNev << teljesSzam << teljesCim << "\n";
    }

    /// Virtuális destruktor
    virtual ~Szemely() {}
};

#endif
```

cim.h

```
#ifndef CIM_H
#define CIM_H

#include "string5.h"
#include "adat.h"
using std::cout;

class Cim : public Adat {
    String orszag;
    String varos;
    String utca;
    String hazszam;
    String iranyitoszam;

public:
    /// Konstruktor beállítja az attribútumokat
    /// @param o - ország megnevezése - String típusú, alapértelmezett üres string
    /// @param v - város megnevezése - String típusú, alapértelmezett üres string
    /// @param u - utca megnevezése - String típusú, alapértelmezett üres string
    /// @param h - házszám megnevezése - String típusú, alapértelmezett üres string
    /// @param i - irányítószám megnevezése - String típusú, alapértelmezett üres string
    Cim(String o = "", String v = "", String u = "", String h = "", String i = "")
        : orszag(o), varos(v), utca(u), hazszam(h), iranyitoszam(i) {}

    /// Város lekérdezése
    String getVaros() const { return varos; }

    /// Város beállítása
    /// @param varos - város értéke
    void setVaros(const String& varos) { this->varos = varos; }
```

```
    /// Ország lekérdezése
    String getOrszag() const { return orszag; }

    /// Ország beállítása
    /// @param orszag - ország értéke
    void setOrszag(const String& orszag) { this->orszag = orszag; }

    /// Utca lekérdezése
    String getUtca() const { return utca; }

    /// Utca beállítása
    /// @param utca - utca értéke
    void setUtca(const String& utca) { this->utca = utca; }

    /// Házszám lekérdezése
    String getHazszam() const { return hazszam; }

    /// Házszám beállítása
    /// @param hazszam - házszám értéke
    void setHazszam(const String& hazszam) { this->hazszam = hazszam; }

    /// Irányítószám lekérdezése
    String getIranyitoszam() const { return iranyitoszam; }

    /// Irányítószám beállítása
    /// @param iranyitoszam - irányítószám értéke
    void setIranyitoszam(const String& iranyitoszam) { this->iranyitoszam = iranyitoszam; }

    /// Cím objektum beolvasása cin-rõl
    Cim cim_beolvas(){
        String o,i,v,u,h;
        std::cout << "Orszag: ";
        std::cin >> o;
        std::cout << "Iranyitoszam: ";
        std::cin >> i;
        std::cout << "Varos: ";
        std::cin >> v;
        std::cout << "Utca: ";
        std::cin >> u;
        std::cout << "Hazszam: ";
        std::cin >> h;
        return Cim(o,v,u,h,i);
    }
    /// Virtuális destruktor
    virtual ~ Cim(){}
};

/// Kiíratás operátor
std::ostream& operator<<(std::ostream& os,const Cim& c);


#endif
```

nev.h

```
#ifndef NEV_H
#define NEV_H

#include "string5.h"
#include "adat.h"
using std::cout;
using std::cin;

class Nev : public Adat {
    String keresztnev;
    String vezeteknev;

public:
    /// Konstruktor beállítja az attribútumokat
    /// @param v - vezetéknév megnevezése - String típusú, alapértelmezett üres string
    /// @param k - keresztnév megnevezése - String típusú, alapértelmezett üres string
    Nev(String v = "", String k = "") : keresztnev(k), vezeteknev(v) {}

    /// Keresztnév lekérdezése
    String getKeresztnev() const { return keresztnev; }

    /// Vezetéknév lekérdezése
    String getVezetektnev() const { return vezeteknev; }

    /// Keresztnév beállítása
    /// @param keresztnev - keresztnév értéke
    void setKeresztnev(const String& keresztnev) {
        this->keresztnev = keresztnev;
    }

    /// Vezetéknév beállítása
    /// @param vezeteknev - vezetéknév értéke
    void setVezeteknev(const String& vezeteknev) {
```

```
        this->vezeteknev = vezeteknev;
    }

    /// Adat kiírása ostream-re
    std::ostream& kiir(std::ostream& os) const {
        return os << vezeteknev << " " << keresztnev << "\t";
    }

    /// Nev objektum beolvasása cin-rõl
    Nev nev_beolvas() {
        String v, k;
        cout << "Vezeteknev: ";
        cin >> v;
        cout << "Keresztnev: ";
        cin >> k;
        return Nev(v, k);
    }

    /// Virtuális destruktor
    virtual ~Nev() {}
};

/// Kiíratás operátor
std::ostream& operator<<(std::ostream& os, const Nev& n);

#endif
```

szam.h

```
#ifndef SZAM_H
#define SZAM_H

#include "string5.h"
#include "adat.h"
using std::cout;

class Szam : public Adat {
    String privat;
    String munkahelyi;

public:
    /// Konstruktor beállítja az attribútumokat
    /// @param p - privát szám megnevezése - String típusú, alapértelmezett üres string
    /// @param m - munkahelyi szám megnevezése - String típusú, alapértelmezett üres string
    Szam(String p = "", String m = "") : privat(p), munkahelyi(m) {}

    /// Munkahelyi szám lekérdezése
    String getMunkahelyi() const { return munkahelyi; }

    /// Privát szám lekérdezése
    String getPrivat() const { return privat; }

    /// Adat kiírása ostream-re
    std::ostream& kiir(std::ostream& os) const {
        return os << privat << " " << munkahelyi << "\t";
    }

    /// Munkahelyi szám beállítása
    /// @param munkahelyi - munkahelyi szám értéke
    void setMunkahelyi(const String& munkahelyi) {
        this->munkahelyi = munkahelyi;
    }

    /// Privát szám beállítása
    /// @param privat - privát szám értéke
    void setPrivat(const String& privat) {
        this->privat = privat;
    }

    /// Szám objektum beolvasása cin-rõl
    Szam szam_beolvas() {
        String m, p;
        std::cout << "Maganszam: ";
        std::cin >> p;
        std::cout << "Munkahelyi: ";
        std::cin >> m;

        return Szam(p, m);
    }

    /// Virtuális destruktor
    virtual ~Szam() {}
};

/// Kiíratás operátor
std::ostream& operator<<(std::ostream& os, const Szam& s);
```

```
#endif
```

adat.h

```
#ifndef ADAT_H
#define ADAT_H

#include "string5.h"
using std::cout;

class Adat {
public:
    /// Virtuális függvény, amely felül lesz írva az ebbõl származtatott osztályok függvényével
    virtual void kiir() const {}

    /// Virtuális destruktor
    virtual ~Adat() {}
};

#endif
```