Házi feladat
Telefonkönyv

Programozás Alapjai 2
Feladatváasztá/feladatspecifikáció

Csutár Márk Tibor
LMVWRE
2022. április 26.

1. Feladat

Telefonkönyv

Tervezze meg egy telefonkönyv egyszerűsített objektummodeljjét, majd valósítsa az meg! A Telefonkönyben 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 törtlése
- Listázás

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

2. Pontosított feladatspecifikáció

A program célja:

A telefonkönyv egy parancssoros menüvezérelt program, mely egy telefonkönyvnek megfelelően személyek adatait és telefonszámait tárolja, valamint azok kezelését teszi lehetővé.

A program használata:

A program az indítása után a parancssorra kiírja a főmenüt, egy számozott listaként. Az egyes almenük és opciók kiválasztásához a listaelem számát kell beírni az alapértelmezett bemenetre.

A menüben található menüpontok:

- Személy adatainak felvétele
- a tárolt adatok listázása
- valamely személy törlése
- személy keresése
- egy személy adatainak módosítása

Elvárt formátumok:

A programba bevitt adatok a megfelelő tárolásért az alábbi feltételeket kell teljesítség:

- név: szöveg
- becenév: szöveg
- cím: szöveg
- munkahelyi szám: 9 jegyű pozitív egész szám
- privát szám: 9 jegyű pozitív egész szám

Ahhoz, hogy egy személy adatai tárolhatóak legyenek, a névre kötelezően szükség van, a többi mező maradhat üresen is.

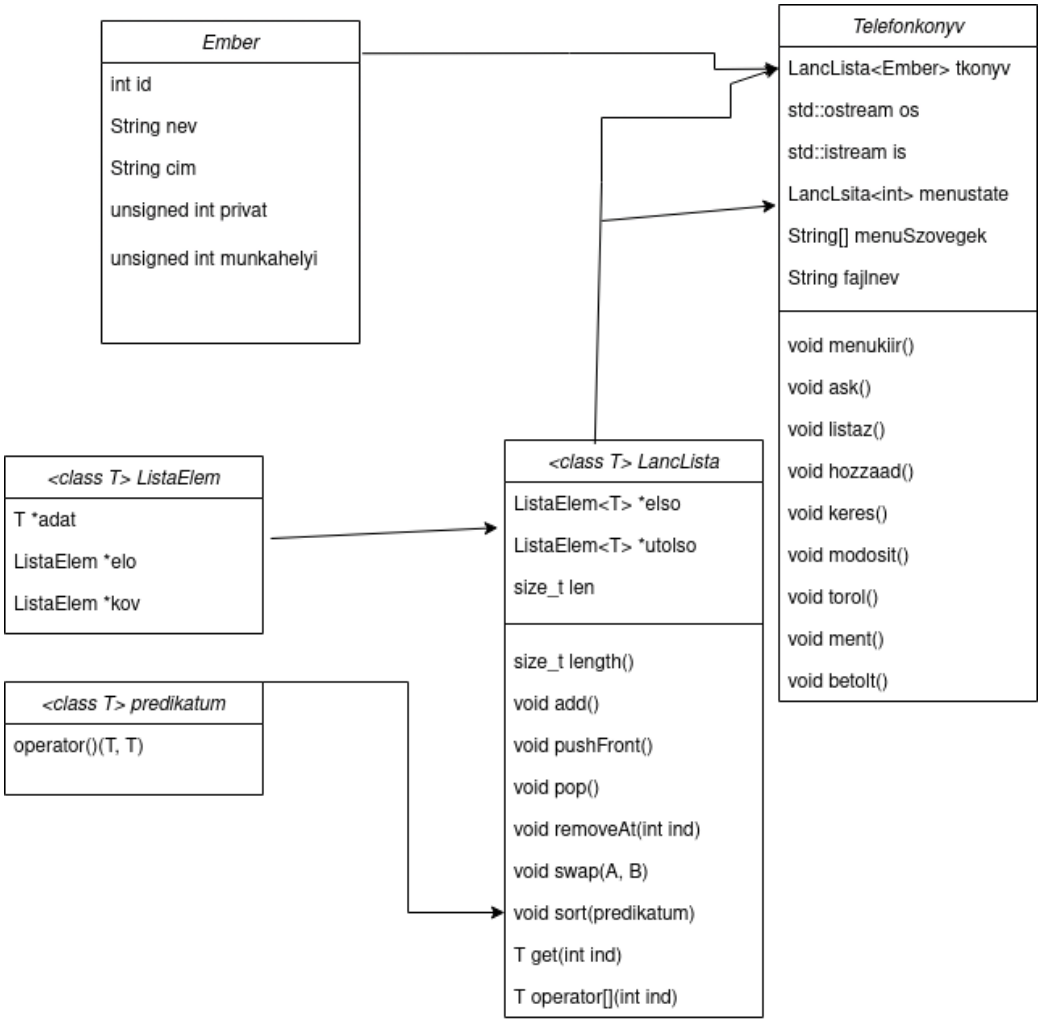
A program továbbá minden személynél generál egy sorszámot, ami egyedi azonosítóként fog szolgálni, valódi sorrendiségre nem utal.

Tesztelés:

A teszteléshez egy olyan programot írok, ami a telefonkönyv funkcióira létrehozott függvényeket és osztályokat, különféle mintaadatokkal kipróbálja.

3. Terv

Osztálydiagram



Osztályok és funkcióik

- Ember: *A programban trárol személyeket leíró osztály. Minden személynek van neve, címe, munkahelyi és privát telefonszáma. Hogy minden Ember egyedi legyen a telefonkönyvön belül, egy Ember rendelkezik egy id-val is, amit a program generál, míg a többi adatot a felhasználó biztosítja.*
- ListaElem: *Sablonos struktúra, egy láncolt lista egy elemét írja le*
- Lanclista: Sablonos *LáncoltLista adatszerkezet, dinamikusan tud nőni, indeyelhető. A telefonkönyv a Emberek tárolására fogja használni.*
  - *Függvények:*
    - *add(), pushFront(), insertAt(): ezek a függvények adatot szúrnak be a láncolt listába*
    - *pop(), popFront(), removeAt(): adatot törölnek a listából*
    - *get(), operator[]: adott indexnél levő elemet adják vissza.* ***Fontos: a listában memória szerint össze-vissza vannak az adatok, az indexelés csak egy kényelmi funkció, nagy listákban nem hatékony módszer egy elem megtalálására.***
    - *sort(pred): Sorbarendezi a listát a megadott predikátumnak megfelelően.*

- Telefonkonyv: Ö*sszefogó osztálya a többi osztálynak, mondhatni „interface”. A telefonkönyv osztály tartalmaz példányokat a többi osztályokból, valamint ő végzi az adatok kiírását és bekérését egy be -és kimenetre, amik mibenlétét szintén magába foglalja. Ez az osztály felel a fájlba írásért és beolvasásért is, amiben az emberek adatai eltárolódnak.*
  - Változók, *az osztályban tároljuk:*
    - *a ki és bemeneti fajl nevét*
    - *egy String tömböt ami adott menüben kiíródik*
    - *mutatókat a ki és bemenetre (ahová írunk és bekérünk a felhasználóval kommunikálva)*
    - *Egész számokat tároló láncolt listát. Ez nem fog nagyra nőni, a szerepe annak a tárolása, a felhasználó mely menüben melyik opciót választotta*
    - *Egy Ember típusó objektumokat tároló láncolt listát, ez maga a telefonkönyvben tárolt adatok.*
  - Függvények: Ezen osztály függvényei alapvetően a felhasználóval kommunikálnak, majd annak megfelőlen kezelik a többi osztály függvényeit
    - *hozzaad(): A tárolt bementről bekéri egy új ember adatait*
    - *listaz(): kiír egy szép listát az összes személy tárolt adatával a kimenetre*
    - *kereses(): Bekéri a keresési szempontot, ahoz egy kulcsut, majd a szintén megadott sorrenben kilistázza a keresésnek megfelelő tárolt személyeket.*
    - *torol(): Bekéri egy személy sorszámát az egyértelmeű azonosítás végett, és kitörli a listából*
    - *ask(), menuKiir(): ask bekéri a felhasználótól egész szám formáhában a választott menüpontot. A menuKiir pedig ennek megfelelően, kiírja a menponthoz eltrátol szöveget, ez alapesetben a főmenü lesz.*
    - *mentes(), betoltes(): hogy a program későbbi kilépéskor és újraindításkor ne veszítse el a ttárolt személyek adatait, fájlba menti azokat. A tárolt fájlnevű fájlba a mentes és a betoltes fuggvenyek végzik el a fájlkezelést.*
- Predikatum: *Struktúra, csak egy () operátorral. Az ősosztály virtuális, örökléssel specifikálható különvöző esetekre. Két érték viszonyáról tér vissza logikai értékkel. A láncolt lista rendezésekor használjuk*

Egyéb algoritmusok

- **bool abcbe(String a, String b, bool eq):** Eldönti, hogy két szting abc rendben van, vagy egyezést mutat-e. Ha eq paraméter hamis, igaz értéket ad vissza, amennyiben a-t hátrébb kell tennni (tehát nincs abc rendben), hamisat akkor, ha a két paraméter abc rendben van. Eq esetén azt nézi meg, egyeznek a karakterek betüi. Fontos: a függvény csak a rövidebb string hozzáig vizsgálja mindkét paraméter Stringet, kis és nagybetűkre érzéketlen, ez szándékos, így változatosan használható a keresésre és rendezésre.

4. Dokumentáció

lanclista.h

- **bool abcbe(String a, String b, bool eq):** Eldönti, hogy két szting abc rendben van, vagy egyezést mutat-e. Ha eq paraméter hamis, igaz értéket ad vissza, amennyiben a-t hátrébb kell tennni (tehát nincs abc rendben), hamisat akkor, ha a két paraméter abc rendben van. Eq esetén azt nézi meg, egyeznek a karakterek betüi. Fontos: a függvény csak a rövidebb string hozzáig vizsgálja mindkét paraméter Stringet, kis és nagybetűkre érzéketlen, ez szándékos, így változatosan használható a keresésre és rendezésre.
- **Template <class T> rescueListaElem(ListaElem<T> le):** A paraméterként kapott elem előtti és utáni listaelemek a paraméter elemre mutató attribútumait, a paraméter elem előtti és utáni elemekre láncolja.
- **Template <class T> ListaElem:** A láncolt lista elemit definiáló adatstruktúra, tárol egy sablon T típusú adat, valamint cím szerinti hivatkozást az előtte és utána levő elemre.
- **Template <typename T> predikatum:** Absztract struktúra, () operátor 2db T típusú objektum viszonyáról ad vissza logikai értéket. Örökléssel specializálható és példányosítható.
- **Template <class T> LancLista:** Láncolt lista adatszerkezet akármennyi T típusú adat tárolására. Csak paraméter nélkül hívható konstuktora van, ami inicializálja, adatot később lehet hozzáadni. Függvényei:
  - **erease**(): *törtli a lista tartalmát.*
  - **length():** *size_t-ben visszaadja, hány elem van jelenleg a listában tárolva*
  - **getElso/Utolso():** *ListaElem<T> mutatót ad a lista első vagy utolsó elemére*
  - **add(const T& d/ListaElem<T> *LE):** d adatot, vagy LE listaelemet a lista végére fűzi.
  - **insertAt(int idx, const T& d/ListaElem<T> *LE):** *A listában helyet fogaló idx-edik helyre befűzi az LE elemet, vagy egy d adatot tartalmazó új listaelemet.*
  - **pushFront(const T& d/ListaElem<T> *LE):** *d adatú új, vagy LE listaelemt a lista elejére fűzi.*
  - **removeAt(int idx):** *idx-ik elem törlése a listából*
  - **pop/popFront():** *Első vagy utolsó elem törlése a listából*
  - **swap(ListaElem<T> *a, ListaElem<T> *b):** *a és b helyet cserél a listában*
  - **sort(predikatum pred. bool asc):** *Predikátum szerint „növekvő” sorba rendezi a listát - ha asc igaz-, „csökkenőbe”, ha hamis. Buborékrendezéssel.*
  - **get(int idx):** *ListaElem mutatót ad a lista idx-ik elemére.*
  - **getData(int idx)/operator[int idx]:** *idx-ik elem adatának rederenciáját adja vissza.*

ember.h

- **Ember:** Egyszerű adattároló osztály egy, a telefonkönyvben tárolni kívánt személy adatait írja le.
  Változók:
  - String nev, becenev, cím: a személy neve, beceneve és címe
  - unsigned int privat, munakaleyi: telefonszámok
  - unsigned int sorszam: az ember azonosító sorszáma

  Függvényei:
  - **Ember(uns. int id, String nev, becenev, cím, uns. int munkahelyi, privat):** paraméteres konstruktor, egyben paraméter nélkül is hívható. A megadott adatokal pldányosítja az osztályt.
  - **Ember(Ember&):** *Másoló konstruktor*
  - **get...():** *Visszaadja a nevet, becenevet, címet, telefonszámok valamelyikét vagy a sorszámot.*
  - **set...(ertek):** *Átállítja az osztályban tárolt értéket a névnek/becenévnek/címnek… a paraméterként kapott értékre. Egyedül a sorszámnak nincs settere.*

- **operator==(const Ember& rhs):** *Összehasonlít 2 embert, igazat ad, ha a tárolt adatok mindegyike azonos.*
- **pperator=(const Ember& rhs):** Értékadó oprátor, rhs ember példány értékei átmásolódnak és felülírják a tároltakat.

telefonkonyv.h

- **enum MENUK:** A menü-almenü szintek számosítva
- **enum FOMENU:** A főmenüből választható menüpontok számai
- **enum SORREND:** Az sorrendet igénylő almenük menüpontjai
- **enum NOVEKVO:** A sorrendes almenük után a növekvő/csökkenő sorrend számértékei
- **const String[] szovgk:** A menük-almenük alap szövegeinek konstansa. A telefonkönyv nem kell kötelezően ezt hordozza magában, lehet akár idegen nyelvű menü szövege is.
- **Telefonkonyv:** A program átfogó osztálya, ami egyben a kommunikációért is felel az adattárolók és a felhasználó között.

  Változók:
  - LancLista<Ember> tkonyv: a telefonkönyvben személyek adatait tároló láncolt lista
  - LancLista<int> menuState: a programot vezérlő menüben választtott menüpontokat tároló láncolt lista
  - const String *menuSzovegek: Adott menüben kiírando szövegek tömbje
  - String fajlnev: a tárolt személyek adatainak fájlba mentéséhez használt fájl neve
  - std::ostream *os: Kimenet, ahová megejelnítjük a menük és a máveletek eredményét
  - std::istream *is: Bbemenet, ahová a felhasználó megadja a bekárt adatokat
  - unsigned int nextId: A legközelebb hozzáadott személy sorszáma

  Függvények:
  - **Telefonkonyv():** *Paraméter nélkül hívható konstruktor. Ilyenkor az alapfájlnév „szemelyek.txt”, a menuSzovegek a szovgk konstans, ki és bemenet pedik std::cout és std::cin*
  - **Telefonkonyv(String fn, const String *msz, std::ostream *o, std::istream *i):** *Paraméteres konstruktor, a fajlnevet, menüszöveget, ki- és bemenetet paraméteren lehet megadni. A nextId itt is 1-ről indul.*
  - **menuKiir():** *A manuState-ben tárolt választásoknak megfelelő menüt írja a kimenetre.*
  - **bool ask()***: Bekér egy számor a bemenetről, ami ha az elvárásnak megfelel, eltárolja a menuState listában. Igaz értékkel tér vissza, ha a választással a felhasználó még nem lép ki a programból.*
  - **listaz(const LancLista<Ember>&):** *A parméterként kapott embereket tároló lista minden elemének minden adatát táblázatszerűen a kimenetre írja.*
  - **Ember beker():** *A bementről bekéri egy újonnan hozzáadni kívánt személy adatait, és azt visszaadja.*
  - **hozzaad(const Ember&):** *A paraméterként kapott Ember objektumot hozzáfűzi a tkonyv láncolt lista végére*
  - **modosit():** *A bementrol bekeri egy személy sorszámát, az egyértelmá azonosítás végett. Az adott sorszámú Ember-t megkeresi a tkonyv listában, ha van találat akkor kitroli az elemet a listabol, ha nincs, a kimenetre irja, hogy nincs adott sorszamu ember.*
  - **kereses():** *A menüben választott attribútom, és keresési kulcs alapján kilistázza azokat a tárolt személyeket, akiknél a választott attribútom hasonlóságot mutat a keresési kulcsra. Ez nem kell hogy pontos egyezés legyen, még számok esetén sem.*
  - **torol():** *A bementrol bekeri egy személy sorszamat, az egyértelmű azonosíthatóság végett. Ha van személy a tárolt sorszámmal, akkor azt törli a tkonyv listabol, ha nincs, azt kimeneten közli a felhasználóval.*

- **mentes(), betolt():** *A tárolt fájlnevű fájl megpróbálja megnyitni és abba beleírni, vagy onnan beolvasni és Ember formájúvá törve az adatokat eltárolni a fájl tartalmát. Ha a fájl nem létezik, mentéskor létrehozzuk, betöltéskor pedig fájltartalom helyett üressen tartjuk a tkonyv listát. Ugyanígy kiürítjük a tkonyv listát, van fájl, de egy sora hibás. A mentés felülírja a fájl korábbi tartalmát az újjal, a betoltes pedig a programban tárolt adatokat írja felül.*
- **get...():** *Csak teszteléshez releváns. Visszaadja a tkonyv vagy menuState lista, vagy a nextId referenciáját.*

5. Működés

Ahhoz, hogy az osztályaink összeálljanak egy menüvezérelt programmá, nincs més teendőnk, mint a main() függvényben példányosítani a Telefonkonyv osztályt, paraméteresen vagy anélkül.

A példánynak ezután a menuKiir() függvényével azonnal kimenetre írható a főmenü, amiből az ask() függvény meghívásával már választhati a felhasználó. Mivel ezen a 2 függvényen a választástól függően mehgívódnak az osztály többi függvényei is, elég ezt a kettőt ismételve meghívni, mindaddig, míg végül a kilépést nem választja a felhasználó. Az ask() logikai visszatérési értéke révén ez legegyszerűbben egy hátultesztelős cikklussal megvalósítható.

A főprogramom ezen felül tartalmaz teszteket is lentebbi menüpontban részletezve.

4. Tesztelés

A tesztelés a főprogramban történik. A főprogram összesen tartalmazza magát a programot, azonnal példányosítva a Telefonkonyv osztályt és ciklikusan meghívva a vezérlő függvényeit. Lokálisan futtatva akár el is lehet ezzel játszani, a Jporta-ra azonban mellékeltem egy bemeneti fájlt, ami tartalmát átirányítva a program bemenetére minden telefonkönyves funkció meghívódik, úgy, hogy az ne okozzon kilépés után változást a szemelyeket tároló, szemelyek.txt-ben (Bár ennek ellenkezője sem okozna gondot, itt most azért történ így a megvalósítás, hogy újra futtatás esetén is ugyanazt az eredményt kapjuk, újra-újra futtatgatás esetén pedig, hogy ne nőjön a szemelyek.txt).

A telefonkönyvből való kilépés után a főpropram tartalmaz további teszteseteket, melyeket a gtest-el valósítottam meg. Ezek külön esetekre bontva példányosítják, és meghívják a függvényeit a program elkészítéséhez használt minden osztálynak, ellenőrízve, hogy azok adott bemenet esetén megfelelő eredménnyel térnek-e vissza.