Házi Feladat

Programozás Alapjai 2.

Feladatválasztás/feladatspecifikáció

Sajben Dániel

VX4USX

2023.04.15

---

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. Specifikáció:

Feladatomban egy egyszerűbb telefonkönyvet kívánok megvalósítani. A minimum kötelező műveleteken kívül még szeretnék beletenni plusz műveleteket is. Ezek a következők:

- Meglévő adatok szerkesztése
- Nevek szerint ABC-sorrendbe rendezés
- Plusz az alapok:
    - adatok felvétele
    - adatok törlése
    - listázás

A telefonkönyvet a felhasználó tudja majd feltölteni, de a megfelelő működés demonstrálása érdekében 5db próba személlyel feltöltöm én is a könyvet, melyeket később a felhasználó természetesen fog tudni törölni.

Ha a felhasználó plusz névjegyet kíván hozzáadni a következő adatokat kéri a program bemenetként, attól függően, hogy céget vagy embert kíván beszúrni.

- Név (vezetéknév, keresztnév)
- becenév
- cím
- privát szám
- Adószám

A felhasználó név alapján tud keresni, ilyenkor a bemenet egy név és a program kimenetként a személy/cég adatait adja.

Lesz olyan lehetőség is hogy listázás, ilyenkor nincs bemenet, hanem a program az összes névjegyet felsorolja egymás után, ABC-sorrendben. Ez akkor hasznos, ha a felhasználó nem emlékszik pontosan a keresett személy teljes nevére.

Az alap, automatikusan létrejövő tagfügvényeket is meg fogom egyénileg valósítani, a program megfelelő működésének érdekében, melyek a következők:

- másolás
- értékadás
- létrehozás
- megszüntetés

A program hibás bemenetek esetén kivételeket fog dobni, ilyen esetek a következők, mint pl.:

- Telefonszám helyére betű kerül
- A felhasználó nem töltötte ki az összes kért adat mezőt
- A felhasználó olyan nevet keres, amely nem található a könyvben.

A teszteléshez is fogok készíteni egy tesztprogramot, mely létre fog hozni egy új névjegyet, keresni fog a névjegyek között, szerkeszteni fogja az egyik névjegy egyik adattagját, törölni fog egy névjegyet, ezután ABC-sorrendben kilistázza az összes névjegyet, és végül leteszteli, hogy a program jól működik-e hibás bemenet esetén is.

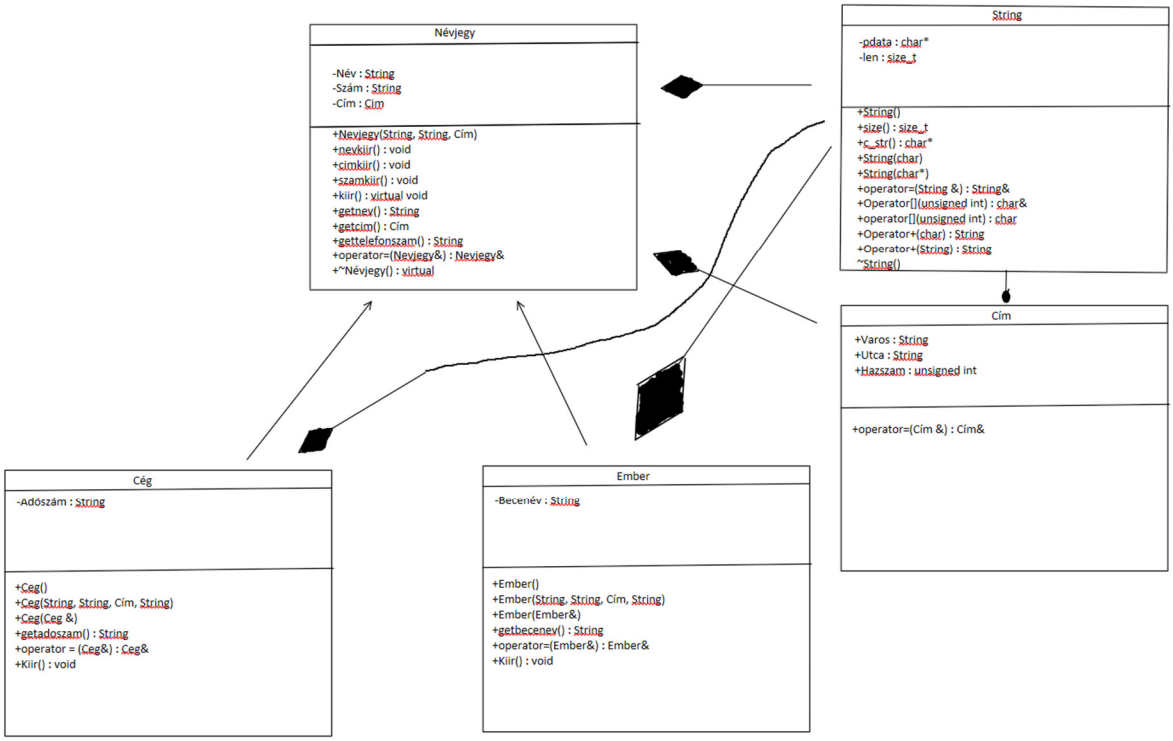
Ezen kívül nyilván a program minden részét lehet manuálisan is tesztelni.

-2023.05.04 - 2.specifikácio - (Nagy Házi TERV) -

Az új részeket, sárgával kiemeltem az előző részben.

Készítettem egy osztálydiagrammot, melyen jól látszanak az objektumaim, és az ezek között lévő kapcsolatok. Az ábrát itt lent látják. Lényegében vagy egy Névjegy ősosztályunk, ilyen típusú pointereket fogunk tárolni egy tömbben, a Telefonkönyv nevű heterogén kollekciót megvalósító osztályban, mely az ábrán nincs jelölve, az egyszerűbb átláthatóság érdekében, ezt itt írom le. A tömbben lévő névjegyek lehetnek, Emberek vagy cégek hiszen ezek alosztályai a Névjegy ősosztályunknak. A

közös függvényeket az ősosztály virtuális függvényként valósítja meg a helyes működés érdekében. A String osztályt minden osztály tartalmazza, ez a laboron elkészített String5 osztályhoz nagyon hasonlóan működik. A cím osztály pedig lényegében egy struktúra, amit az ősosztály használ, ezért a kódban ez a Névjegy.h fájlban lesz deklarálva. A funkciók nagy része, ami az ábrán nem látszik, az a mainben lesz megvalósítva.



-2023.05.12 - 3.specifikácio - (Skeleton) -

A házifeladatnak ennek a részében tudom, hogy még csak a deklarációkat kellett volna megvalósítani, viszont az én programom ennél már előrébb tart, remélem ez nem okoz problémát. A program minden funkciója már működő képes, minden függvény és osztály teljes mértékben elkészült. Már a memtrace is „include”-olva van és elméletileg nincsen memória szivárgás sem. A végső program azonos lesz a most feltöltött programmal, annyi különbséggel, hogy most még a tesztelés manuális, mert a „gettest” még hiányzik a programból. Ez még megvalósításra vár. A feltöltött fájlok közül hiányzik a .txt fájl, ami a program működéséhez elengedhetetlen, mert abból olvas ki a program és abba ment, viszont a „skeleton” feladatrésznél azt írták, hogy csak .h,.cpp,.hpp,.pdf formátumú fájlokat töltsünk fel és ez .txt, ezért döntöttem úgy, hogy ezt még nem töltöm fel.

-2023.05.27 - 4.specifikácio - Végleges –

A program most már teljes mértékben elkészült. Ebben a specifikáció részben, főként a program és a függvények működését fogom leírni, a tervezéshez kapcsolódó információkat, és az UML ábrát az előző részek már tartalmazzák.
Összesen 5 db osztályom van (String, Nevjegy, Ceg, Ember, Telefonkonyv), ezekből a String az a laboron elkészített osztály, ezért ennek a függvényeire és működésére nem térnék most ki. A Stringet minden osztály tartalmazza mert az adattagok String típusúak. Az ősosztályunk a „Nevjegy” ebből származtattam a „Ceg” és az „Ember” osztályt, mert a „Telefonkonyv”-ben ezt a 2db típusú névjegyet lehet tárolni. A Telefonkonyv osztály, az egy heterogén kollekciót valósít meg, ebben tároljuk a névjegyekre mutató pointereket. A Névjegy, a Cég és az Ember osztályok nagyon hasonló függvényeket tartalmaznak, ilyenek a „get()” függvények, a konstruktorok, destruktorok, az „=” operátor illetve a „kiir()” függvények, melyekről egyszerűségük miatt, szerintem nem kell sokat beszélni. A Névjegy osztály egy absztrakt osztály, hisz vannak virtuális függvényei, amelyek a leszármazottakban is megjelennek, ilyen függvények a „kiir()” és a „getadat()” függvények, illetve a destruktorok. A komolyabb függvényeket a Telefonkönyv osztály tartalmazza. Ezeket most egyesével bemutatom, mert ezekkel dolgozik a programunk, tehát megértésük elengedhetetlen.

Konstruktor:
```
Telefonkonyv() : size(0), Tomb(nullptr) {}
```
- A „size” adattagot 0-ra állítja és a tárolót pedig egy nullpointerre inicializálja.

```
void fileba_ment(const char* filename);
```
- Egy filenevet kér be az argumentumba, ami egy .txt file kell, hogy legyen a megfelelő működéshez, és ebbe a fileba elmenti a tárolt névjegyeket. (Hibát is kezel)

```
void keres (String name)
```
- Ez a függvény bemenetként egy nevet kér (String típust), amit ezután megkeres a tárolóban (ha nincs benne, hibát dob) és erre az elemre meghívja ennek a „kiir()” függvényét és ezzel kiírja a kimenetre a névjegyet, de visszatérési értéke nincsen.

```
void szerkeszt(String nev, String szerkesztett, char mit)
```
- A szerkeszt függvény bemenetként kér egy nevet, hogy kit szeretnénk szerkeszteni, hogy melyik adatot szeretnénk szerkeszteni, és, hogy mire szeretnénk változtatni ezt. Ez a függvény is hibát dob, ha nincs ilyen ember, akit szeretnénk szerkeszteni. A függvény a végén meghívja a rendez() függvényt, ami azért fontos mert, ha esetleg a nevet szerkesztjük, akkor lehet, hogy ABC sorrendben a tarolóban később kell, hogy megjelenjen ez a névjegy a jövőben. Visszatérési értéke ennek a függvénynek sincsen.

```
template<class T>
void Insert(T* kontakt);
```
- Az Insert függvény egy „template” függvény, amire nem lenne feltétlenül szükség, mert ebben az esetben csak névjegy típusú elemeket tárolunk, és a Cég és az Ember típus is ebből van származtatva, tehát ha argumentumként Névjegy pointert kérnénk be úgy is jól működne, de a „template”-es megoldást, elegánsabbnak éreztem és így a jövőben minimális módosításokkal más helyen is tökéletesen lehet majd használni ezt a függvényt.

```
void filebol_feltolt(const char* filename)
```
- Ez a függvény arra való, hogy a .txt fileból feltöltse a tárolónkat, így tudjunk az elmentett adatokkal dolgozni. Ez a függvény is egy filenevet kér be, és ebből a fileból feltölti a tárolót. Hibát is kezel, ha esetleg nem tudjuk megnyitni a file-t, visszatérési értéke ennek a függvénynek sincsen.

```
void rendez()
```
- A rendez függvény egy visszatérési érték nélküli függvény, ami csak annyit csinál, hogy a tároló elemeit, név alapján „ABC” sorrendbe rendezi.

```
void elem_torlese(String name)
```

- Az elem_torlese függvény bemenetként egy nevet kér, hogy kit szeretnénk törölni a tárolóból, és ezt az embert eltávolítja. Hibát is kezel, ha esetleg nincs ilyen illető a tárolóban.

```
void listaz()
```

- A listáz függvény csak kiírja az elemeket a standard outputra, úgy, hogy végigmegy a tömbön és minden elemre meghívja a hozzá tartozó „kiir()” függvényt.

```
void torol()
```

- Ez függvény kiüríti a teljes tárolót, úgy hogy minden elemet egyesével kitöröl, majd a tömböt is. A „size”-ot nullára állítja és a tömböt pedig egy nullpointerre.

```
Nevjegy* operator[] (size_t x) {return Tomb[x];}
```

- Szerintem az „[]” operátort nem kell részleteznem, szimplán visszaadja a tároló x-edik elemét.

```
Destrukotor: ~Telefonkonyv()
```

- A destruktor meghívja a „torol()” függvényt, ezzel kitörli az összes elemet.

Beszéljünk egy kicsit a main.cpp-ről is, hiszen ez valósítja meg magát a programot. Itt már új függvény nem készül, csak az eddig felsoroltakat használom. Igazából egy nagyon egyszerű „interface” van, amivel lehet kezelni a programot. Az opciók közötti választást pedig egy „switch-case” mechanikával oldottam meg, mert szerintem így volt a legátláthatóbb. Törekedtem arra, hogy a programban az összes függvényt használjam és ezáltal minden, függvény működésének a tesztelése könnyen megoldható legyen. Írtam is egy test.txt, amivel tesztelem a programot. A program mindig felhívja, a felhasználó figyelmét arra, hogy mi a helyes bemenet formátuma, hogy a program megfelelően működhessen, igaz törekedtem arra, hogy helytelen bemenet esetén a program kilépjen és azt ne mentse el.

A .txt-ben alapból 3 adat van, a tesztelés érdekében, az első sorban azt tárolom, hogy hány db névjegy van és alatta minden sorba egy-egy névjegyet. Ez fontos, hogy ha valaki saját .txt file-t akar használni abba a formátum ugyan így legyen megvalósítva.