**házi feladat**

Italrecept

Végleges

Osvárt Bence

FDYUGK

2020. május 16.

**TARTALOM**

[1. Feladat - Italrecept **Hiba! A könyvjelző nem létezik.**](#_Toc289114914)

[2. Pontosított feladatspecifikáció 2](#_Toc289114915)

[3. Terv 2](#_Toc289114916)

[3.1. Objektum terv 2](#_Toc289114917)

[3.2. Osztályok dokumentációja 3](#_Toc289114918)

[3.3. Main függvények 5](#_Toc289114921)

[3.4. Interfész – Felhasználói dokumentáció 7](#_Toc289114922)

# Feladat – Italrecept

Készítsen C++ programot italreceptek nyilvántartására!

A program alapanyagokból építse fel a recepteket. Elegendő 3-4 receptet kezelnie, de legyen bővíthető. A programmal minimálisan a következő feladatokat kell ellátni:

• új alapanyag bevitele az adatbázisba

• új recept bevitele az adatbázisba • adatbázis kiírása fájlba

• adatbázis beolvasása fájlból

• recept törlése

• kiválasztott ital hozzávalóinak listázása

Egyszerű felhasználói felületet tervezzen!

A feladat lényege az objektumorientált megközelítés, ill. modellezés és nem a felhasználói felület szépsége. Használjon heterogén adatszerkezetet! A megoldáshoz ne használjon STL tárolót!

# Pontosított feladatspecifikáció

A feladat nem írja, hogy milyen formában tároljam az adatokat, ezért én a láncolt listát választottam.

A feladathoz még adtam egy-két extra tartalmat, név szerint a listának van neve, hozzávalói és ami az extra az, hogy van elkészítési leírása és elkészítési ideje is minden receptnek.

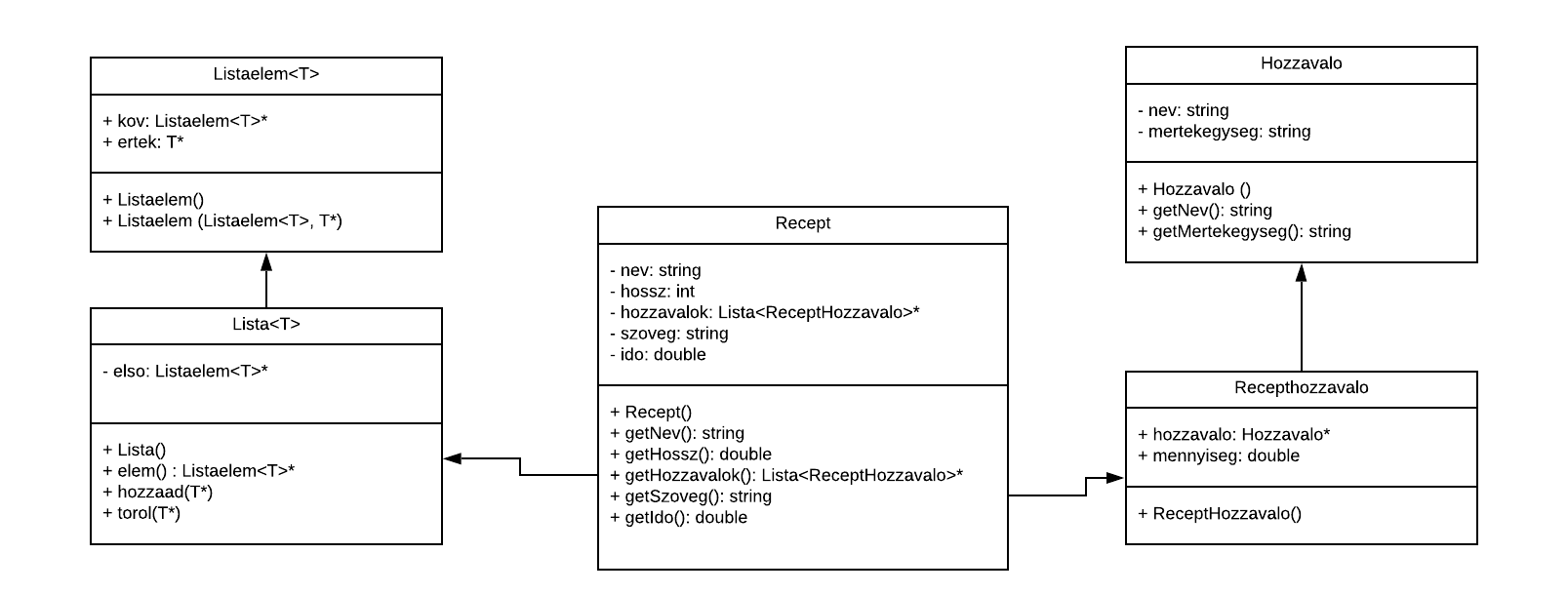
Viszont a fájl beolvasás csak a megfelelő formátummal működik, azaz ahogy korábban elmenti. (3.2-nél részletesen taglalva lesz.)

A tesztekkel a főbb függvényeket ellenőrzöm, hogy jól működjenek.

# Terv

A feladat két osztály és egy láncolt lista osztály megtervezését igényli.

## Objektum terv



Az ellenőrzéshez a *memtrace.cpp* és *memtrace.h* fájlokat használtam.

## Osztályok dokumentációja

**Listaelem**

Egy láncolt lista egyik eleme.

Publikus attribútumok:

kov: Listaelem<T>\* A kovetkező elemre mutató pointer

ertek: T\* A pointer értéke

Tagfüggvények:

Listaelem (T\* ertek) Konstruktor

Listaelem (Listaelem<T>\* elozo, T\* ertek) Konstruktor két elem összefűzésére

(a T\* lesz a következő elem)

**Lista**

Az adatok kezelésére láncolt lista sablont csináltam. A sablon sablonparaméterként veszi át az első elemet és a következő elemre mutató pointert. Új elem hozzáadásához és elem törlésére külön függvényt csináltam.

Privát attribútum:

elso: Listaelem<T>\* A Lista első eleme

Tagfüggvények:

Lista() Konstruktor

elem(): Listaelem<T>\* Visszatér a lista első elemével

hozzaad(T\* ertek): void Hozzáad egy elemet a láncolt lista végéhez

torol(T\* ertek): void Kitöröl egy elemet a láncolt lista akármelyik

pontjából

~Lista() Destruktor

**Hozzavalo**

Az egyik fő osztály aminek egy eleme egy hozzávalót és a mértékegységét tartalmazza.

Privát attribútumok:

nev: string Egy hozzávakó neve

mertekegyseg: string Egy hozzávaló mértékegysége

Tagfüggvények:

Hozzavalo(std::string nev, std::string mertekegyseg)

A konstruktor

getNev(): string Visszatér az elem privát névvel

getMertekegyseg(): string Visszatér az elem métékegységével

(sidenote: Ezt az osztályt valószínűleg csak láncolt listában használom, azért nincs több függvénye)

**Recepthozzavalo**

Ez csak egy átmeneti osztály, azokra az esetekre amikor a következő (Recept) osztály hozzávalói mellé egy mennyiség is kerül.

Nyilvános attribútumok:

hozzavalo: Hozzavalo\* Egy hozzávaló ami mellé mennyiséget adunk

mennyiseg: double Egy mennyiség.

Tagfüggvények:

ReceptHozzavalo(Hozzavalo\* hozzavalo, double mennyiseg)

A konstruktor.

operator<<(std::ostream& lhs, ReceptHozzavalo& rhs): ostream&

Kiír operátor

A receptben a felsoroláshoz létrehoz egy "sor" elemet,

azaz mennyiség "\_" mért.egys. "\_" hozzávaló "\_" alakban.

**Recept**

A második fő osztály típus, ami egy-egy teljes ital receptjét tárolja el.

Privát attribútumok:

nev: string Eltárolja az ital nevét

hossz:int Eltárolja a hozzávalóinak számát

hozzavalok: Lista<ReceptHozzavalo>\* Egy láncolt lista, amiben a hozzávalói vannak

az italnak.

szoveg: string Egy elkészítési útmutató, avagy leírás.

ido: double Az ital elkészítéséhez való idő.

Tagfüggvények:

Recept (…) Sok meghívott taggal a konstruktor.

getNev(): string Visszetér az ital nevével

getHossz():int Visszatér az ital hozzávalóinak hosszával

getHozzavalok(): Lista<ReceptHozzavalo>\* stb…

getSzoveg(): string …

getIdo(): double …

~Recept() Destruktor

operator<<(std::ostream& lhs, Recept& rhs): ostream& Kiír operátor

Kiírja a recept nevét, alá a hozzávalóit (soronként egyet)

Az elkészítésének módját es az elkészítési időt.

## Main függvények

### Fájlba mentés – fájlból olvasás

A feladat egyetlen összetett algoritmusa a beolvas() és a ment() függvények, legalábbis számomra.

A legnagyobb macera az volt vele, hogy minden információ el legyen mentve és megfelelő sorrendben, majd úgy legyen visszaolvasva a következő program indításnál.

Amivel majdnem egy hétig szenvedtem az a beolvasásnál egy apró hiba, hogy egy láncolt listába olvastam be a külön receptek hozzávalóit és emiatt rosszul is mentette el minden futás után… Szerencse, hogy megtaláltam.

beolvas(Lista<Hozzavalo>& hozzavalok, Lista<Recept>& receptek): void

A fejlécben referenciaként megkapott listákat tölti fel a megadott txt fájlokból

ment(Lista<Hozzavalo>& hozzavalok, Lista<Recept>& receptek): void

A fejlécben megkapott listák minden adatát lementi a megadott 2 txt fájlba.

### Hibakezelés

Az esetek nagyrészében az adott helyen lokálisan csináltam whlie ciklusokat, hogy amíg nem veszi fel a bemenet a megfelelő értéket, addig újat kér a program a felhasználótól.

Viszont kaptam egy kódrészt, amit template függvénnyé alakítottam és a beadás előttig mindig működött is, de valamit megunt és valamiért nem mindig akar rendesen működni.

Itt a függvény kódrészlete:

template <typename T> T readT(std::istream& lhs) {

T result;

if (!(lhs >> result)) {

lhs.clear();

result = -1;

//lhs.ignore(32767, '\n');

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cout << "Nem jo a bemenet! ";

}

std::string skip;

std::getline(lhs, skip);

return result;

}

### A többi menüpont és maga a menü

A többi függvény fejlécet leírom és röviden elmondom a működést, mert a kód meg a kommentek magukért beszélnek.

menu(): void

Ez a függvény amíg a felhasználó használja a programot rendre kiíródik, de semmilyen adatot nem kér be maga a függvény csak szöveget ír ki.

Kiir\_all(Lista<Recept>& rhs): void

Ez a függvény kiír minden receptnek a Nevét számozott listában.

Recept\_kiir((Lista<Recept>& receptek): void

Egy kiválasztott recept minden részét kiírja a recept kiír konstruktorral, ha a recept sorszámát beírod (a fentebbi listabeli számot a neve elől)

UjRecept(Lista<Hozzavalo>& hozzavalok, Lista<Recept>& receptek): void

Step by step látre lehet hozni egy új ital receptjét viszont visszalépni és kilépni nem lehet közben, és egy megadott adat után, azt javítani se. (még)

Recept\_torol(Lista<Hozzavalo>& hozzavalok, Lista<Recept>& receptek): void

Ki lehet törölni egy receptet, annak a sorszámával együtt.

### Tesztprogram

3 tesztem van, ami a struktúrák létrejöttéért felel test1-2-3 rendre, és mellette még a fájlból olvasás és mentésnek vannak tesztjei, mert a program többi része könnyedén, hiba mentesen meglett elsőre és azoknak úgy éreztem nem kell ellenőrzés.

(mindegyik teszt az *NagyHF-italrecept.cpp* fájlban található)

/// Hozzávaló létrejötte

void test\_1() {

Hozzavalo\* test = new Hozzavalo("Víz","dl");

if (strcmp(test->getNev().c\_str(), "Víz") != 0)

throw std::exception();

if (strcmp(test->getMertekegyseg().c\_str(), "dl") != 0)

throw std::exception();

}

A test függvényekben vannak a hiba dobások (throw) és a testek main részében vannak a catch-ek amik elkapják a hibákat ha van.

try {

int nr;

nr = readT<int>(std::cin); // hányadik teszteset

switch (nr) {...}

// kivétel elkapása

}

catch (std::exception& e) {

std::cerr << e.what() << std::endl;

return 2;

}

catch (...) {

std::cerr << "mas hiba";

return 3;

}

return 0;

## Interfész – Felhasználói dokumentáció

A felhasználói oldal, azaz amikor a program fut.

Amikor elindul a program kiírja a menüt, amit minden parancs befejezése után újra kiír meg újra.

* Itt a felhasználó a látott listából választhat egy parancsot és annak a számával elkezdheti azt. (1- 4 közötti szám, de szöveget nem tud mindig kezelni 3.3.3.)

(Sajnálatos módon, ha a felhasználó kiválaszt egy parancsot, azt végig kell csinálni, mert nem lehet félbeszakítani egyiket se.)

* Az egyes menüpont kiírja a meglévő italok nevét.
* A kettes menüpontban az egyik ital sorszámával ki lehet írni annak az italnak a receptjét.
* A hármas menüpontban létrehozhat a felhasználó egy új (vagy már meglévő receptet) De sorban kell haladni a lépésekkel.

Először a recept neve, majd a hozzávalói egyesével

Majd a recept elkészítési útmutatója röviden és az ital elkészítési idejét kell megadni.

* A négyes menüpont az egy recept törlése az adatbázisból, ehhez a listában szerepló italnév előtti sorszámot kell beírni miután kiválasztottuk ezt a parancsot és kitörli a receptet ha ki tudja és kiírja a program, hogy sikeres volt-e.
* Az ötös pont az a kilépés, azt a 404 szám beírásával teheti meg, ahogy a menü is mutatja, ilyenkor lesz minden változtatás az adatbázisban elmentve szintén.