Zombori Péter (DSYU1P)

# Nagyházi specifikáció

## Feladatleírás

A feladat a Sportegyesület feladat. A program egy sportegyesület csapatait tartja nyilván. Három féle csapat lehet, Kézilabda csapat, Kosárlabda csapat és Focilabda csapat. Minden csapatnak vannak közös tulajdonságaik, például a nevük és a csapatok létszáma. Minden csapatnak vannak sajátos tulajdonságaik. A kézilabda csapatnak jegyzi az éves támogatásait, a kosárlabda csapat jegyzi a pompomlányok számát, a focicsapat jegyzi az edzőik számát.

## Adatstruktúra

A feladat adatstruktúrája több osztályra van bontva. A csapat osztály a szülőkönyvtár, ahol a közös tulajdonságok jelennek meg. Illetve van a három örökös osztály a saját tulajdonságaikkal. Az adatok a csapat típusa *txt* fileban vannak tárolva. Ez 3db file: *kezi.txt*, *kosar.txt*, *foci.txt*. A fileokból beolvasott adatok egy heterogén láncolt listában tárolódnak. Azért jobb ehhez a feladattípushoz a láncolt lista, mint a tömb, mert a sok csapat esetén egy csapat törlése nagyon sok erőforrást venne igénybe (csinálni egy egyel rövidebb tömböt és bemásolni mindent), míg a láncolt listánál csak összekötöm az előtte és következőre mutatót. Ugyan ez igaz új csapat felvételére.

Fontos, hogy a program esetleges bővítésénél jól felhasználható legyen, ezért van a Nyilvántartás osztály, amely egy API-ként szolgál, egybefogja az adattípusok funkcióit, beszédes elnevezésű funkciókkal rendelkezik. Ez által a *gtest_lite* segítségével könnyedén lehet az adatokat tesztelni.

Fontos továbbá, hogy a felhasználó számára is jól nézzen ki a program. Mivel a program parancssoros program, ezért a körülményeknek megfelelően átgondolt designra van szükség. A felhasználó egy menürendszer segítségével fog tudni navigálni a funkciók között. Minden almenü egy tabulátorral beljebbről kezdődik, hogy átlátható legyen a menürendszerben való navigálás. Ezt egy Menü osztály valósítja meg, amelyen, ha meghívjuk a főMenü függvényt, 'átveszi az uralmat' a program felett, és a menürendszer lekezelését fogja megvalósítani. A menürendszer lekezeli a fileok beolvasását és az adatok fileokba lementését, és a felhasználóval való kommunikációt. Beolvasni elég a program indulásakor, menteni pedig automatikusan adatmódosuláskor, vagy programból való kilépéskor kell.

A csapatok nevei, láncolt lista hossza nem tudottak, ezeket a program dinamikusan kezeli. Általában nincs megkötés semmilyen input hosszában, kivéve a konzol sztenderd inputról való olvasásnál, ami maximum 256 karakter lehet. Ez nem befolyásolja az átlag felhasználót, mert nem életszerű egy 257 karakter hosszú csapatnév. Az adatok dinamikus kezelésénél fontos a memóriaszivárgás elkerülése, amiben a *memtrace* segít.

## Dokumentáció

A dokumentációt automatikusan a Doxygen a GraphViz és a MiKTeX generálja a *header* fileokban a függvények, osztályok, struktúrák felett elhelyezett Doxygen kommentekből. Egy sok oldalas *Docs.pdf* PDF dokumentum és egy interaktív weboldal a dokumentáció. A weboldal, a *html* mappában az *index.html* fileból indul. A két dokumentáció között óriási különbség nincs. A PDF verzió részletesebb egy fokkal, a HTML verzió interaktívabb. A preferált fő dokumentáció a PDF verzió, a több részlet a hossz miatt és a kompaktsága miatt, de a HTML verzió intuitívabb.

# Menü

A menürendszer fő feleadata a felhasználó és az adatok közti kommunikáció. Képes a következő feladatokra:

A főmenüben:

- Az összes adat kilistázására, csapattípusok almenüjébe navigálni

Az almenükben:

- Az adott csapattípusú csapatokat kilistázni. Csapatnév szerint csapatra keresni, és a keresett csapat almenüjére navigálni.
- Új csapatot beszúrni

A keresett csapat almenüjében:

- A csapat adatait (szintén almenüben módosítani)
- A csapatot törölni

A menük, almenük között számokkal (négynél nem nagyobb számokkal) lehet navigálni. A hibás számot a program kezeli és vár a helyes válaszra.

Az ábrákból látszódik, hogy a Menü használata nélkül is működőképes a program, csak meg kell hívni a Nyilvántartás API megfelelő funkcióit. A Nyilvántartás API egy heterogén láncolt listában fogja össze az adatokat, amely listaelemeit a Lista mutatja be.

# DevTools, Platformok

A mivel a program nem platformfüggetlen, ezért szükséges gondolni a különböző operációs rendszerek viselkedésére. Linux és UNIX típusú operációs rendszerek esetén a fileok között található *makefile* segít a program lefordításában, és egy *nagyhazi* futtatandó fileba fordul le a *make* parancs után (erről dokumentáció a *makefile* fileban). Windows alatt a *Makefile_WIN.cmd* file futtatása fog segíteni a hasonló nagyhazi.exe előállításához. A szkriptek által preferált fordító a g++. Avval rendelkeznie kell a számítógépnek.

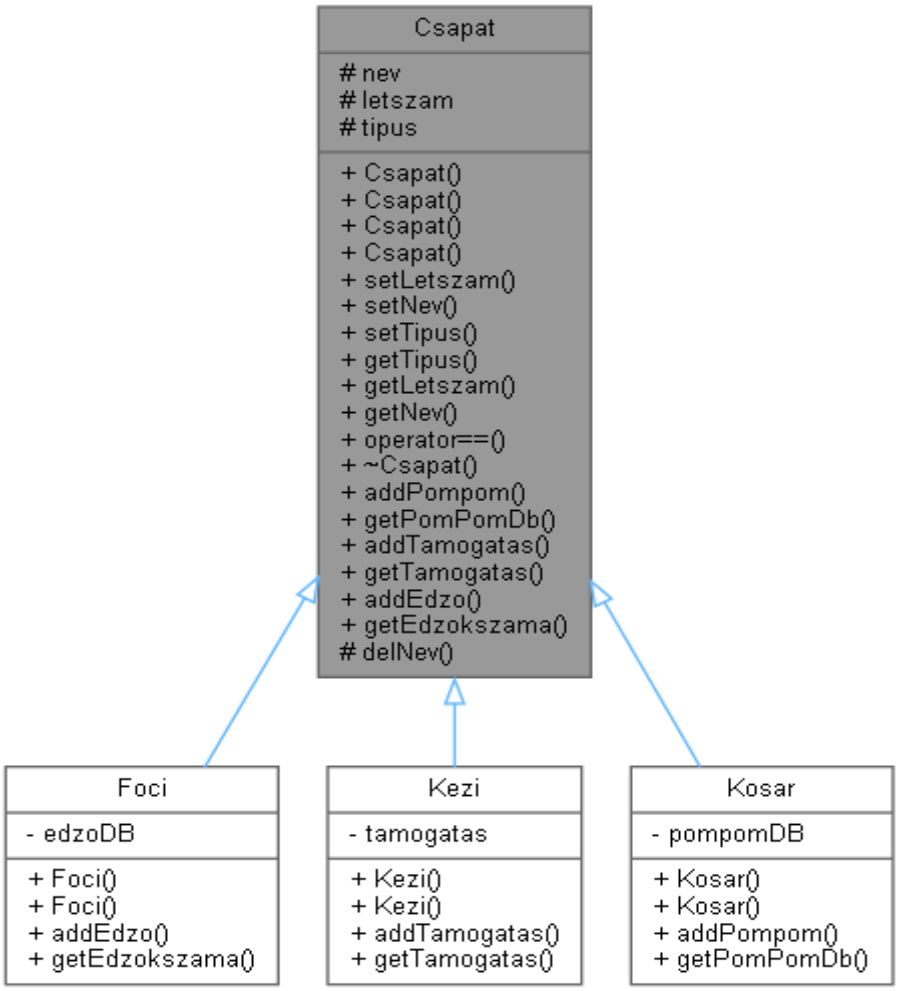
Fontos figyelembe venni a Linux, UNIX, valamint a Windows közti sztenderd input viselkedési különbséget is. Ezt a program forduláskor automatikusan kezeli a preprocesszor, az automatikusan definiálódó operációsrendszer változókkal (*__linux__*, *__APPLE__* stb…)

Nem árt, ha a fordítók közötti külömbséget is figyelembe vesszük, ezért mindegyik fordítási scriptben a fordítón a *--std=c++11* kapcsoló be van állítva, így C++ 11 es verzió alatt fordul le minden program, minden rendszeren.

# Terv

## Adatstruktúra

A heterogén lista építéséhez, szükség van a megfelelő absztrakt objektumokra, öröklődésekre. A csapatok osztálya a heterogén lista működéséhez így néz ki:

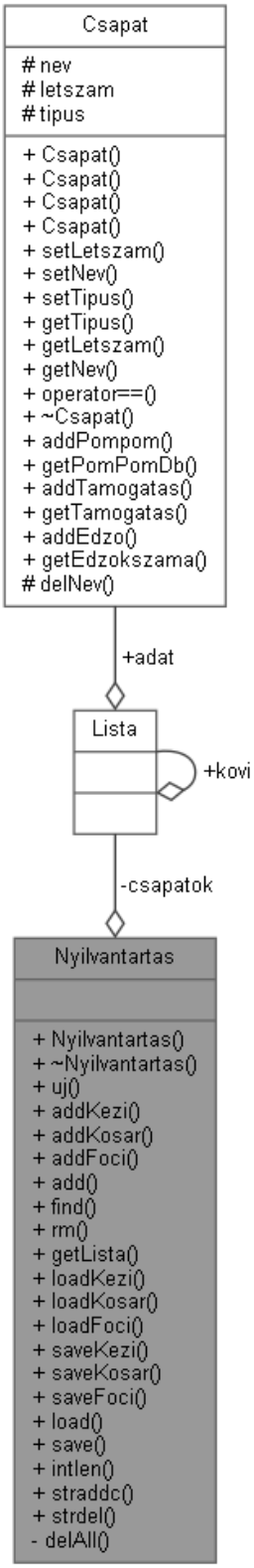



Evvel a konstrukcióval a heterogén lista megvalósítható. Ügyelni kell a nem azonos tulajdonságok lekezelésére, és a típusbiztosságra is. Ezt meg lehet oldani csúnyán kivételek kezelésével, vagy szépen a visszaadási értékek ellenőrzésével. (például, ha egy Kosar csapatra ráhívjuk a getTamogatas() függvényt, az -1-et ad vissza). Erre is nagyon hasznos a típus enum a Csapatban.

# API:

A program teszteléséhez (gtest_lite.h), bővítéséhez, használatához készül egy Nyilvántartás osztály, ami egy API-ként szolgál az adatok alacsony szintű kezelésére. Beszeédes függvénynevekkel, rendelkezik, a megfelelő használhatósághoz. A heterogén láncolt listát a privát 'csapatok' adattagjában tárolja el, a Lista struct megvalósításával (ez valósítja meg a láncolt lista logikát. Ez egy láncolt lista láncszem tulajdonképpen). A 'csapatok' igazából egy Lista pointer.

A Nyilvántartás API működése->




A Nyilvántartásnak egy csomó segédfüggvénye is van, amiket saját maga hív meg, de publikusak az egyszerű használat miatt. (például az intlen(), ami azért kell, mert a Kézilabda csapatok támogatása milliós nagyságrendű lehet, ami túl sok nulla lekezelése, és ezért nem árt tudni, hogy milyen hosszú számról beszélünk….stb)
Ezt az osztályt tesztelve tudhatjuk meg, hogy helyesen működik-e az adatstruktúránk.

# Menü:

A Menü osztály ugyan olyan fontos része a programnak, mint az összes többi adatstruktúra és API, ugyanis ezen keresztül kommunikál a felhasználó az adatokkal.

Fontos, hogy intuitív legyen az egyszerű használhatóság miatt. A program konzol applikáció, úgyhogy nem rendelkezik grafikus interfésszel. A Menürendszer számok bevitelével ellenőrzi a menükben való lépkedést.

A menürendszer DB privát adattagja fogja megvalósítani a nyilvántartást. Ezen kereszül fog kommunikálni a Menü a Nyilvántartás API-val.

A menürendszer foMenu() funkcióját meghívva, a menürendszer 'átveszi az irányítást' a program felett, és elkezdi a program futtatását, például, Nyilvántartás betöltése, felhasználótól válasz kérése, stb…

A menürendszer funkciói egy almenüt szimbolizálnak. A menürendszer ilyen formában lesz implementálva. Itt figyelni, kell a stackoverflow elkerülésére (tehát, ne lehessen egy főmenüt megívni egy almenüből, mert akkor végtelenszer meghívódhatna és a kezdőfeltétel nélküli rekurziós függvények problémája állna fenn) de ilyen nem áll fenn, hiszen a menürendszer logikájából ez következik. (tehát almenü nem hív meg főmenüt.)

A menürendszer pár segédfüggvénnyel is rendelkezik, amik nem almenüket szimbolizálnak. Ilyen például a getStdRowLen(). Ezek a segédfüggvények a kinézet miatt kellenek. Általában tabulátor szám, vagy táblázat szélességének a számolására.

## Fileok, dokumentáció, verziókövetés:

Az osztályok modulárisan fileokra vannak bontva, a jobb átláthatóság miatt. A deklaráció a headaer fájlokban, az implementáció a cpp fájlokban zajlik. Minden függvény, osztály, adattag tulajdonságai, működései a header fájlokban dokumentálva vannak Doxygen kommentek használatával. Ezekből intuitív html weboldal és PDF készül, (Dot, GraphWiz és MikTex segítségével) ahol a program működését látványosan és részletesen át lehet látni.

A verziókövetéshez, változások dokumentálásához GitHub lesz használva.