Nagyhazi

Generated by Doxygen 1.9.6

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

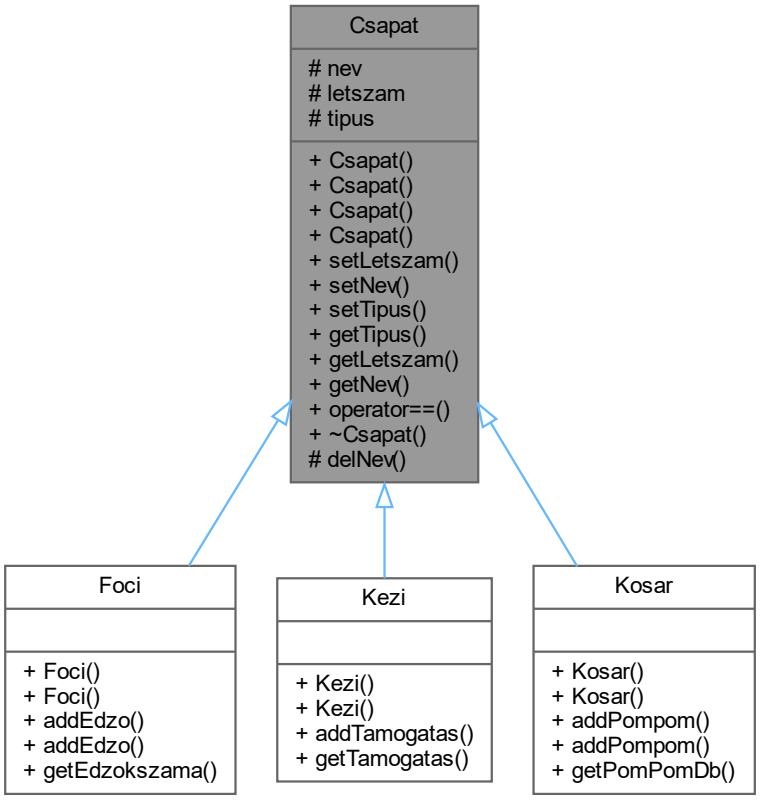
Class Documentation

4.1 Csapat Class Reference

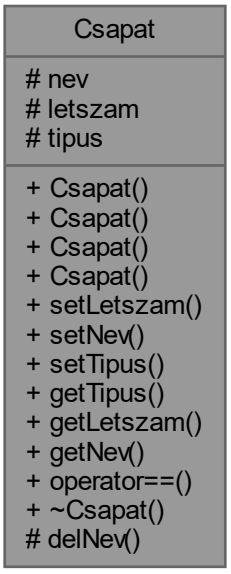
A csapatok szülőobjektuma. Ez írja je a csapatok közös viselkedését, tulajdonságait.

`#include <csapat.h>`

Inheritance diagram for Csapat:



Collaboration diagram for Csapat:



Public Member Functions

- Csapat ()

  Default konstruktor, amely létrehoz egy NINCS típusú, 0 létszámú, semmilyen nevű csapatot.
- Csapat (const char ∗, int)

  Név és létszámos konstruktor, amely létrehozza az adadtoknak megfelelő csapatot.
- Csapat (const char ∗)

  A csapat nevét létrehozó konstruktor, amely az a adoktnak megfelelő csapatot hoz létre, és a létszámot nullázza.
- Csapat (int)

  A csapat létszámot is létrehozó konstruktor, amely nem hoz létre csapatnevet.
- void setLetszam (int)

  A csapat létszámát átállító, beállító függvény.
- void setNev (const char ∗)

  A csapat nevét átállító, beállító függvény.
- void setTipus (const Tipus)

  A csapat típusát beállító függvény (többször átállítani nincs értelme, mert úgyis öröklődik és az örökös tulajdonságai mások).
- Tipus getTipus () const

  Viszaadja a csapat típusát a Tipus enum segítségével.
- int getLetszam () const

  Visszaadja a csapat létszámát.
- const char ∗ getNev () const

  Visszaadja a csapat nevét.
- bool operator== (const char ∗)

  Lehetővé teszi a csapatok közti gyors keresést a == operátor túlterhelésével.
- virtual ∼Csapat ()

  Virtuális destruktor (mert öröklődik majd a class).

Protected Member Functions

- void delNev ()

  Segédfunkció, amely kitörli, felszabadítja a nevet, ha az nem üres.

Protected Attributes

- char * nev

  A csapat neve.
- int letszam

  A csapat létszáma.
- Tipus tipus

  A csapat típusa a Típus enum segítségével.

4.1.1 Detailed Description

A csapatok szülőobjektuma. Ez írja je a csapatok közös viselkedését, tulajdonságait.

Definition at line 23 of file csapat.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Csapat() [1/4]

```
Csapat::Csapat ( )  [inline]
```

Default konstruktor, amely létrehoz egy NINCS típusú, 0 létszámú, semmilyen nevű csapatot.

Definition at line 39 of file csapat.h.

4.1.2.2 Csapat() [2/4]

```
Csapat::Csapat (
            const char * p,
            int n )
```

Név és létszámos konstruktor, amely létrehozza az adadtoknak megfelelő csapatot.

Parameters

csapat_nev	ez a const char* paraméter a csapat neve lesz.
letszam	ez az int paraméter a csapat létszáma lesz.

Definition at line 5 of file csapat.cpp.

4.1.2.3 Csapat() [3/4]

```
Csapat::Csapat (
            const char * p )
```

A csapat nevét létrehozó konstruktor, amely az a adoktnak megfelelő csapatot hoz létre, és a létszámot nullázza.

Parameters

csapat_nev	ez a const char* paraméter a csapat neve lesz.

Definition at line 10 of file csapat.cpp.

4.1.2.4 Csapat() [4/4]

```
Csapat::Csapat (
            int n )
```

A csapat létszámot is létrehozó konstruktor, amely nem hoz létre csapatnevet.

Parameters

letszam	ez az int paraméter a csapatlétszám lesz.

Definition at line 15 of file csapat.cpp.

4.1.2.5 ~Csapat()

```
Csapat::~Csapat ( )  [virtual]
```

Virtuális destruktor (mert öröklődik majd a class).

Definition at line 47 of file csapat.cpp.

Here is the call graph for this function:

4.1.3 Member Function Documentation

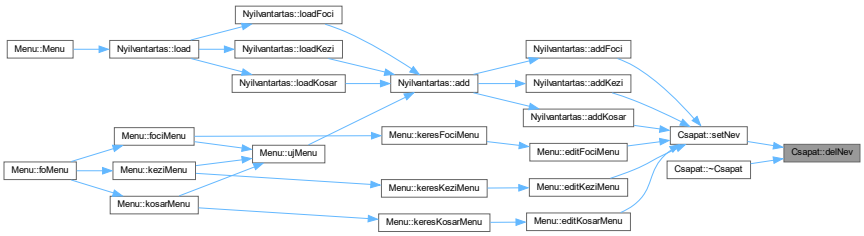
4.1.3.1 delNev()

```
void Csapat::delNev ( )  [inline], [protected]
```

Segédfunkció, amely kitörli, felszabadítja a nevet, ha az nem üres.

Definition at line 35 of file csapat.h.

Here is the caller graph for this function:



4.1.3.2 getLetszam()

```
int Csapat::getLetszam ( ) const
```

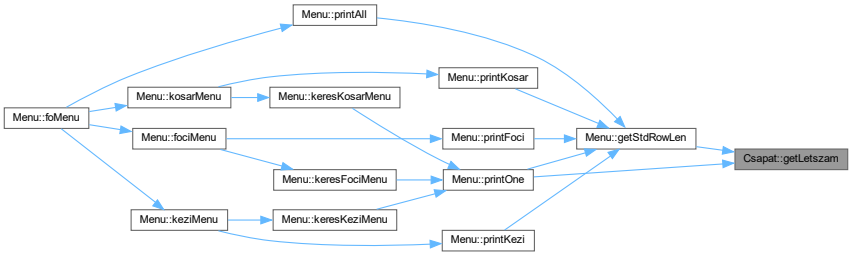
Visszaadja a csapat létszámát.

Returns

A csapat létszáma, int.

Definition at line 35 of file csapat.cpp.

Here is the caller graph for this function:



4.1.3.3 getNev()

```
const char * Csapat::getNev ( ) const
```

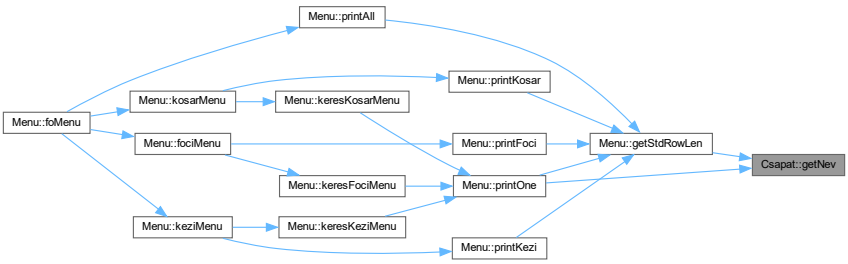
Visszaadja a csapat nevét.

Returns

A csapat neve, const char*.

Definition at line 39 of file csapat.cpp.

Here is the caller graph for this function:



4.1.3.4 getTipus()

```
Tipus Csapat::getTipus ( ) const
```

Viszaadja a csapat típusát a Tipus enum segítségével.

Returns

A csapat típusa a Tipus enum-ban.

Definition at line 31 of file csapat.cpp.

4.1.3.5 operator==()

```
bool Csapat::operator== (
            const char * str )
```

Lehetővé teszi a csapatok közti gyors keresést a == operátor túlterhelésével.

Megvizsgálja, hogy az adott névvel egyezik-e a csapat neve és ennek megfelelő

bool (igaz/hamis) értéket dob vissza.

Parameters

keresett_csapat_nev	A keresett csapatnév const char* paraméter.

Returns

Egy igaz hamis érték, hogy egyezik-e a csapatnév.

Definition at line 43 of file csapat.cpp.

4.1.3.6 setLetszam()

```
void Csapat::setLetszam (
            int n )
```

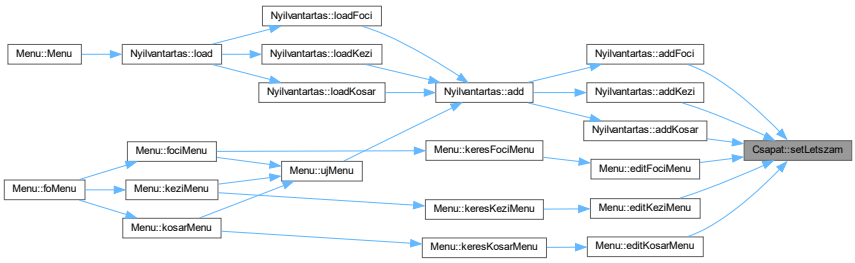
A csapat létszámát átállító, beállító függvény.

Parameters

uj_letszam	Ez az int parameter lesz az új létszáma a csapatnak.

Definition at line 17 of file csapat.cpp.

Here is the caller graph for this function:



4.1.3.7 setNev()

```
void Csapat::setNev (
            const char * p )
```

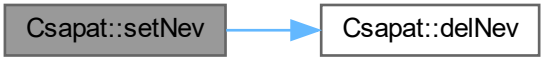
A csapat nevét átállító, beállító függvény.

Parameters

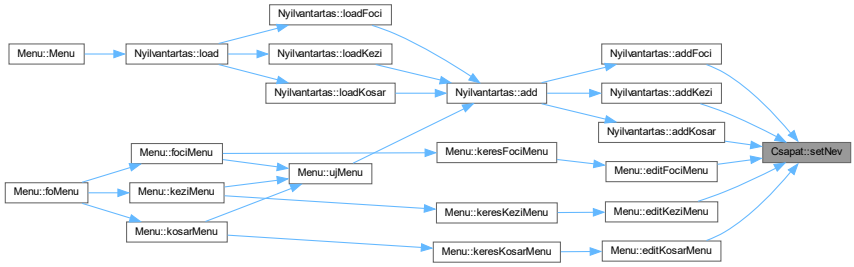
uj_nev	Ez a const char* paraméter lesz a csapat új neve.

Definition at line 21 of file csapat.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.8 setTipus()

```
void Csapat::setTipus (
            const Tipus t )
```

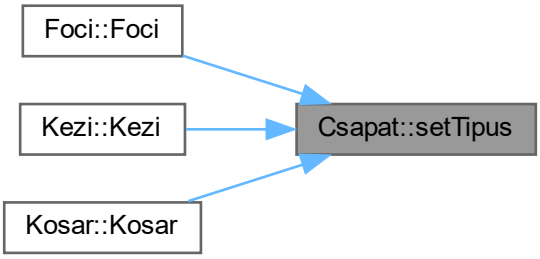
A csapat típusát beállító függvény (többször átállítani nincs értelme, mert úgyis öröklődik és az örökös tulajdonságai mások).

Parameters

uj_tipus	Ez a Tipus parameter a csapat típusát adja meg a Tipus enum segítségével.

Definition at line 27 of file csapat.cpp.

Here is the caller graph for this function:



4.1.4 Member Data Documentation

4.1.4.1 letszam

```
int Csapat::letszam  [protected]
```

A csapat létszáma.

Definition at line 29 of file csapat.h.

4.1.4.2 nev

```
char* Csapat::nev  [protected]
```

A csapat neve.

Definition at line 26 of file csapat.h.

4.1.4.3 tipus

```
Tipus Csapat::tipus  [protected]
```

A csapat típusa a Típus enum segítségével.

Definition at line 32 of file csapat.h.

The documentation for this class was generated from the following files:

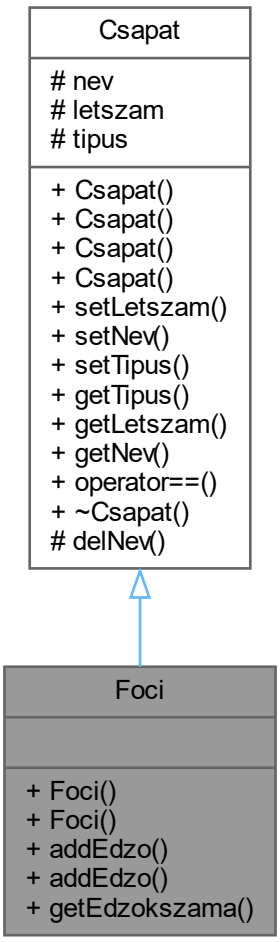
- csapat.h
- csapat.cpp

4.2 Foci Class Reference

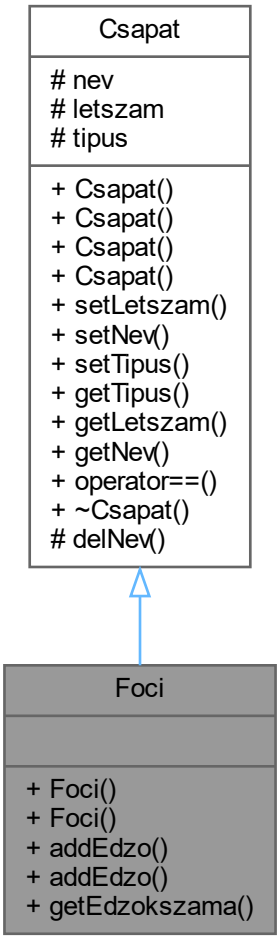
A focicsapat objektumja, amely örökli a Csapat objektum tulajdonságait.

`#include <foci.h>`

Inheritance diagram for Foci:



Collaboration diagram for Foci:



Public Member Functions

- Foci ()

  A default konstruktor, amely a Csapat default konstruktorát használja.
- Foci (const char ∗, const int)

  A konstruktor, amely létrehoz az adatoknak megfelelő Csapat objektumot.
- void addEdzo ()

  Egy darab új edzőt ad hozzá a csapathoz. Növeli az edzoDB countert eggyel.
- void addEdzo (const int)

  Sok darab új edzot ad hozzá a csapathoz. Növeli az edzoDB countert a megfelelő számmal.
- const int getEdzokszama () const

  Visszaadja az edzők számát.

Public Member Functions inherited from Csapat

- Csapat ()

  Default konstruktor, amely létrehoz egy NINCS típusú, 0 létszámú, semmilyen nevű csapatot.
- Csapat (const char *, int)

  Név és létszámos konstruktor, amely létrehozza az adadtoknak megfelelő csapatot.
- Csapat (const char *)

  A csapat nevét létrehozó konstruktor, amely az a adoktnak megfelelő csapatot hoz létre, és a létszámot nullázza.
- Csapat (int)

  A csapat létszámot is létrehozó konstruktor, amely nem hoz létre csapatnevet.
- void setLetszam (int)

  A csapat létszámát átállító, beállító függvény.
- void setNev (const char *)

  A csapat nevét átállító, beállító függvény.
- void setTipus (const Tipus)

  A csapat típusát beállító függvény (többször átállítani nincs értelme, mert úgyis öröklődik és az örökös tulajdonságai mások).
- Tipus getTipus () const

  Viszaadja a csapat típusát a Tipus enum segítségével.
- int getLetszam () const

  Visszaadja a csapat létszámát.
- const char * getNev () const

  Visszaadja a csapat nevét.
- bool operator== (const char *)

  Lehetővé teszi a csapatok közti gyors keresést a == operátor túlterhelésével.
- virtual ~Csapat ()

  Virtuális destruktor (mert öröklődik majd a class).

Additional Inherited Members

Protected Member Functions inherited from Csapat

- void delNev ()

  Segédfunkció, amely kitörli, felszabadítja a nevet, ha az nem üres.

Protected Attributes inherited from Csapat

- char * nev

  A csapat neve.
- int letszam

  A csapat létszáma.
- Tipus tipus

  A csapat típusa a Típus enum segítségével.

4.2.1 Detailed Description

A focicsapat objektumja, amely örökli a Csapat objektum tulajdonságait.

Definition at line 8 of file foci.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Foci() [1/2]

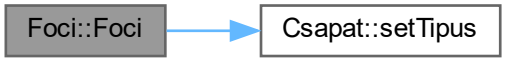
```
Foci::Foci ( )
```

A default konstruktor, amely a Csapat default konstruktorát használja.

tehát létrehoz egy üres csapatot (csak a típusát FOCI-ra rakja).

Definition at line 5 of file foci.cpp.

Here is the call graph for this function:



4.2.2.2 Foci() [2/2]

```
Foci::Foci (
            const char * p = "",
            const int n = 0 )
```

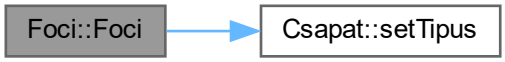
A konstruktor, amely létrehoz az adatoknak megfelelő Csapat objektumot.

Parameters

csapat_nev	a csapat beállítandó neve.
letszam	a csapat beállítandó létszáma.

Definition at line 9 of file foci.cpp.

Here is the call graph for this function:



4.2.3 Member Function Documentation

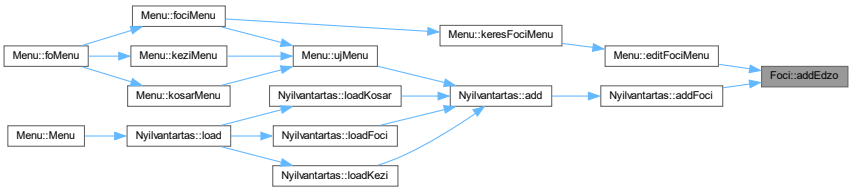
4.2.3.1 addEdzo() [1/2]

```
void Foci::addEdzo ( )
```

Egy darab új edzőt ad hozzá a csapathoz. Növeli az edzoDB countert eggyel.

Definition at line 14 of file foci.cpp.

Here is the caller graph for this function:



4.2.3.2 addEdzo() [2/2]

```
void Foci::addEdzo (
            const int n )
```

Sok darab új edzot ad hozzá a csapathoz. Növeli az edzoDB countert a megfelelő számmal.

Parameters

edzok_szama	ennyivel növeli az edzoDB countert.

Definition at line 16 of file foci.cpp.

4.2.3.3 getEdzokszama()

```
const int Foci::getEdzokszama ( ) const
```

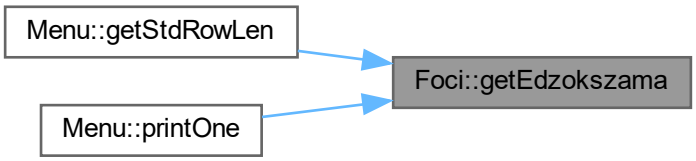
Visszaadja az edzők számát.

Returns

Az edzők száma, int.

Definition at line 18 of file foci.cpp.

Here is the caller graph for this function:



The documentation for this class was generated from the following files:

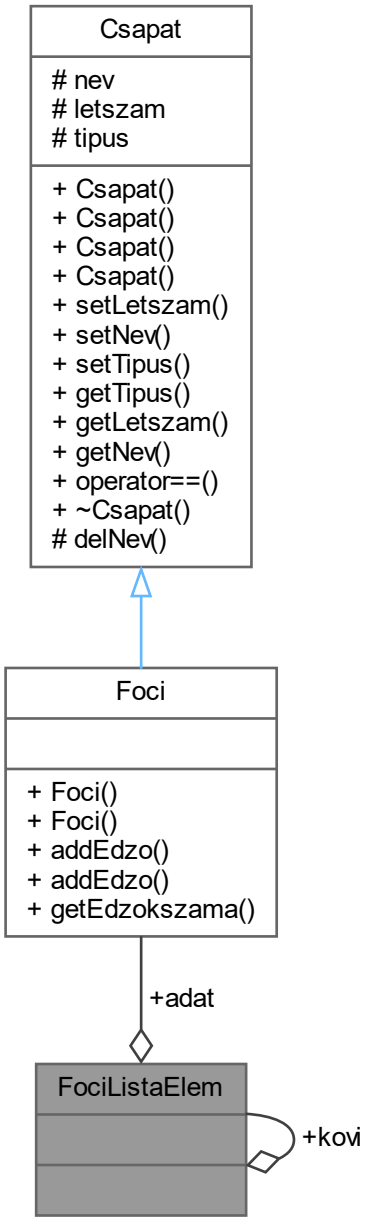
- foci.h
- foci.cpp

4.3 FociListaElem Struct Reference

A Foci osztály láncolt listázásához szolgáló struktúra. Ez egy darab listaelem.

```
#include <nyilvantartas.h>
```

Collaboration diagram for FociListaElem:



Public Attributes

- Foci adat

  A listaelem adata, ami egy Foci osztály.
- FociListaElem * kovi

  A következő listaelemre mutató pointer.

4.3.1 Detailed Description

A Foci osztály láncolt listázásához szolgáló struktúra. Ez egy darab listaelem.

Definition at line 29 of file nyilvantartas.h.

4.3.2 Member Data Documentation

4.3.2.1 adat

```
Foci FociListaElem::adat
```

A listaelem adata, ami egy Foci osztály.

Definition at line 31 of file nyilvantartas.h.

4.3.2.2 kovi

```
FociListaElem* FociListaElem::kovi
```

A következő listaelemre mutató pointer.

Definition at line 34 of file nyilvantartas.h.

The documentation for this struct was generated from the following file:

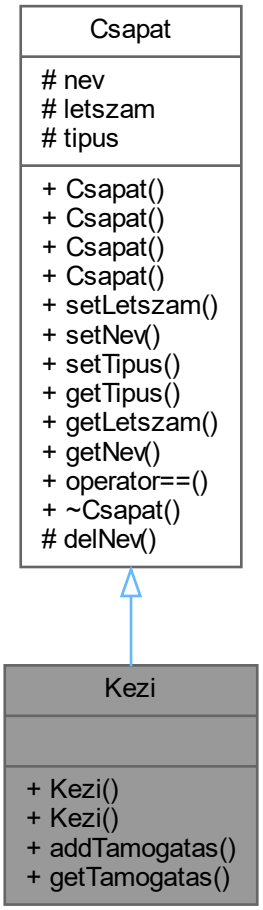
- nyilvantartas.h

4.4 Kezi Class Reference

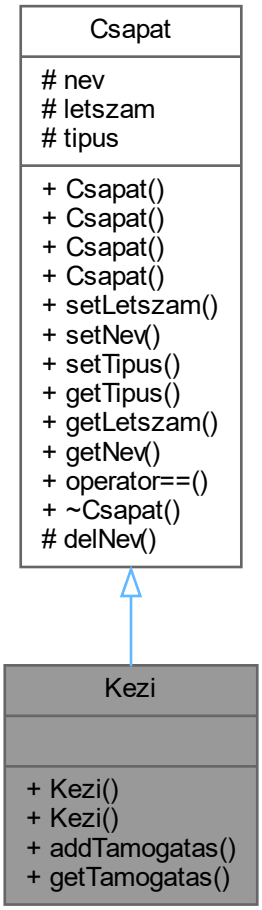
A kézilabda csapat objektumja, amely örökli a Csapat objektum tulajdonságait.

```
#include <kezi.h>
```

Inheritance diagram for Kezi:



Collaboration diagram for Kezi:



Public Member Functions

- Kezi ()

  A default konstruktor, amely a Csapat default konstruktorát hívja, csak a.
- Kezi (const char ∗, const int)

  A konstruktor, amely a megadott adatoknake megfelelően állítja be a csapatot.
- void addTamogatas (const int)

  A megadott számmal növeli a tamogatas változót.
- const int getTamogatas () const

  Visszaadja a támogatások számát.

Public Member Functions inherited from Csapat

- Csapat ()

  Default konstruktor, amely létrehoz egy NINCS típusú, 0 létszámú, semmilyen nevű csapatot.

- Csapat (const char *, int)

  Név és létszámos konstruktor, amely létrehozza az adadtoknak megfelelő csapatot.
- Csapat (const char *)

  A csapat nevét létrehozó konstruktor, amely az a adoktnak megfelelő csapatot hoz létre, és a létszámot nullázza.
- Csapat (int)

  A csapat létszámot is létrehozó konstruktor, amely nem hoz létre csapatnevet.
- void setLetszam (int)

  A csapat létszámát átállító, beállító függvény.
- void setNev (const char *)

  A csapat nevét átállító, beállító függvény.
- void setTipus (const Tipus)

  A csapat típusát beállító függvény (többször átállítani nincs értelme, mert úgyis öröklődik és az örökös tulajdonságai mások).
- Tipus getTipus () const

  Viszaadja a csapat típusát a Tipus enum segítségével.
- int getLetszam () const

  Visszaadja a csapat létszámát.
- const char * getNev () const

  Visszaadja a csapat nevét.
- bool operator== (const char *)

  Lehetővé teszi a csapatok közti gyors keresést a == operátor túlterhelésével.
- virtual ~Csapat ()

  Virtuális destruktor (mert öröklődik majd a class).

Additional Inherited Members

Protected Member Functions inherited from Csapat

- void delNev ()

  Segédfunkció, amely kitörli, felszabadítja a nevet, ha az nem üres.

Protected Attributes inherited from Csapat

- char * nev

  A csapat neve.
- int letszam

  A csapat létszáma.
- Tipus tipus

  A csapat típusa a Típus enum segítségével.

4.4.1 Detailed Description

A kézilabda csapat objektumja, amely örökli a Csapat objektum tulajdonságait.

Definition at line 8 of file kezi.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Kezi() [1/2]

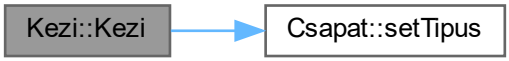
```
Kezi::Kezi ( )
```

A default konstruktor, amely a Csapat default konstruktorát hívja, csak a.

típust KEZI-re rakja a Típus enum segítségével.

Definition at line 5 of file kezi.cpp.

Here is the call graph for this function:



4.4.2.2 Kezi() [2/2]

```
Kezi::Kezi (
            const char * p,
            const int n )
```

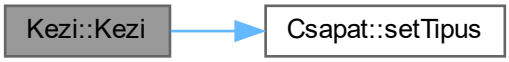
A konstruktor, amely a megadott adatoknake megfelelően állítja be a csapatot.

Parameters

uj_csapat_nev	a csapat uj neve.
letszam	a csapat létszáma.

Definition at line 9 of file kezi.cpp.

Here is the call graph for this function:



4.4.3 Member Function Documentation

4.4.3.1 addTamogatas()

```
void Kezi::addTamogatas (
            const int t )
```

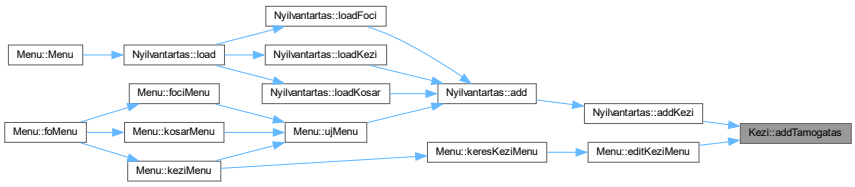
A megadott számmal növeli a tamogatas változót.

Parameters

uj_tamogatas	ennyivel növeli a támogatás változót.

Definition at line 14 of file kezi.cpp.

Here is the caller graph for this function:



4.4.3.2 getTamogatas()

```
const int Kezi::getTamogatas ( ) const
```

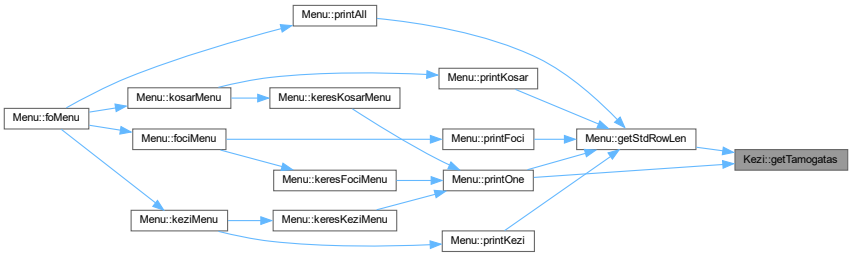
Visszaadja a támogatások számát.

Returns

A támogatások száma.

Definition at line 18 of file kezi.cpp.

Here is the caller graph for this function:



The documentation for this class was generated from the following files:

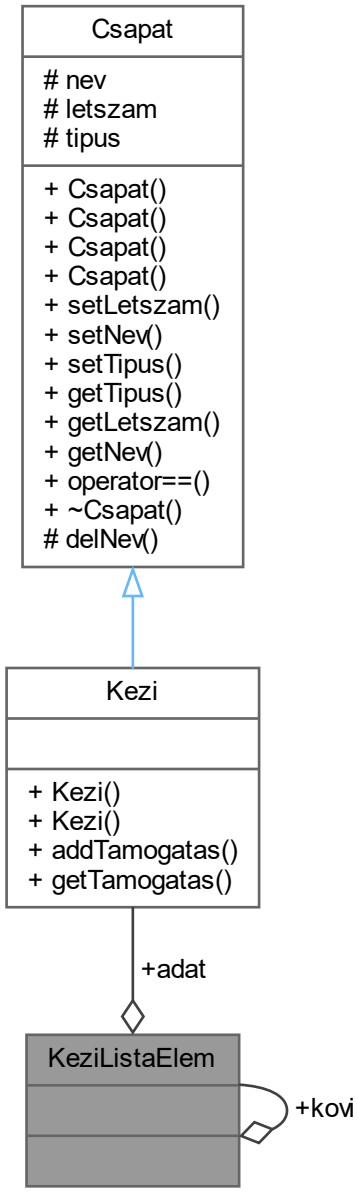
- kezi.h
- kezi.cpp

4.5 KeziListaElem Struct Reference

A Kezi osztály láncolt listázásához szolgáló struktúra. Ez egy darab listaelem.

`#include <nyilvantartas.h>`

Collaboration diagram for KeziListaElem:



Public Attributes

- Kezi adat

  A listaelem adata, ami egy Kezi osztály.
- KeziListaElem * kovi

  A következő listaelemre mutató pointer.

4.5.1 Detailed Description

A Kezi osztály láncolt listázásához szolgáló struktúra. Ez egy darab listaelem.

Definition at line 11 of file nyilvantartas.h.

4.5.2 Member Data Documentation

4.5.2.1 adat

```
Kezi KeziListaElem::adat
```

A listaelem adata, ami egy Kezi osztály.

Definition at line 13 of file nyilvantartas.h.

4.5.2.2 kovi

```
KeziListaElem* KeziListaElem::kovi
```

A következő listaelemre mutató pointer.

Definition at line 16 of file nyilvantartas.h.

The documentation for this struct was generated from the following file:

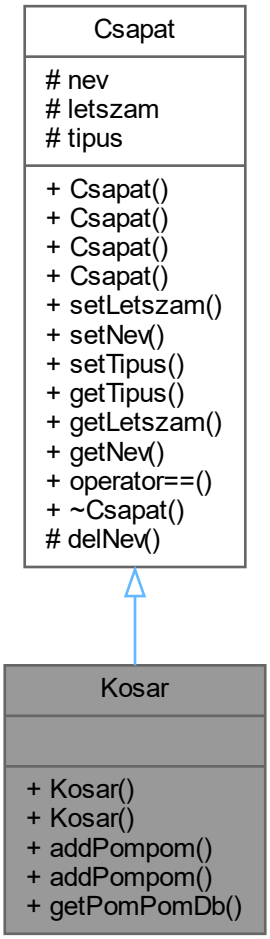
- nyilvantartas.h

4.6 Kosar Class Reference

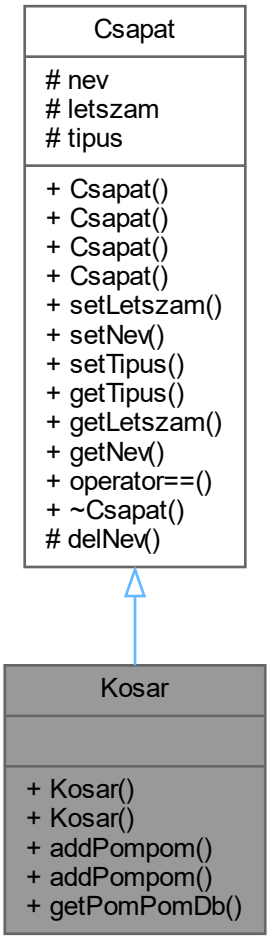
A kosárlabda csapat objektumja, amely örökli a Csapat objektum tulajdonságait.

```
#include <kosar.h>
```

Inheritance diagram for Kosar:



Collaboration diagram for Kosar:



Public Member Functions

- Kosar ()

  A default konstruktor, amely a Csapat default konstruktorát hívja, csak a.
- Kosar (const char ∗, const int)

  Az adatoknak megfelelő csapatot hoz létre.
- void addPompom ()

  Egy darab pompomlányt ad hozzá a csapathoz.
- void addPompom (const int)

  Az adatoknak megfelelő mennyiségű pompomlányt ad a csapathoz.
- const int getPomPomDb () const

  Visszaadja a pompomlányok számát.

Public Member Functions inherited from Csapat

- Csapat ()

  Default konstruktor, amely létrehoz egy NINCS típusú, 0 létszámú, semmilyen nevű csapatot.
- Csapat (const char *, int)

  Név és létszámos konstruktor, amely létrehozza az adadtoknak megfelelő csapatot.
- Csapat (const char *)

  A csapat nevét létrehozó konstruktor, amely az a adoktnak megfelelő csapatot hoz létre, és a létszámot nullázza.
- Csapat (int)

  A csapat létszámot is létrehozó konstruktor, amely nem hoz létre csapatnevet.
- void setLetszam (int)

  A csapat létszámát átállító, beállító függvény.
- void setNev (const char *)

  A csapat nevét átállító, beállító függvény.
- void setTipus (const Tipus)

  A csapat típusát beállító függvény (többször átállítani nincs értelme, mert úgyis öröklődik és az örökös tulajdonságai mások).
- Tipus getTipus () const

  Viszaadja a csapat típusát a Tipus enum segítségével.
- int getLetszam () const

  Visszaadja a csapat létszámát.
- const char * getNev () const

  Visszaadja a csapat nevét.
- bool operator== (const char *)

  Lehetővé teszi a csapatok közti gyors keresést a == operátor túlterhelésével.
- virtual ~Csapat ()

  Virtuális destruktor (mert öröklődik majd a class).

Additional Inherited Members

Protected Member Functions inherited from Csapat

- void delNev ()

  Segédfunkció, amely kitörli, felszabadítja a nevet, ha az nem üres.

Protected Attributes inherited from Csapat

- char * nev

  A csapat neve.
- int letszam

  A csapat létszáma.
- Tipus tipus

  A csapat típusa a Típus enum segítségével.

4.6.1 Detailed Description

A kosárlabda csapat objektumja, amely örökli a Csapat objektum tulajdonságait.

Definition at line 8 of file kosar.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Kosar() [1/2]

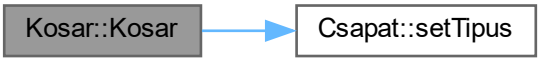
```
Kosar::Kosar ( )
```

A default konstruktor, amely a Csapat default konstruktorát hívja, csak a.

típust átállítja KOSAR-ra a Tipus enum segítségével.

Definition at line 6 of file kosar.cpp.

Here is the call graph for this function:



4.6.2.2 Kosar() [2/2]

```
Kosar::Kosar (
            const char * p = "",
            const int n = 0 )
```

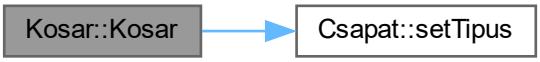
Az adatoknak megfelelő csapatot hoz létre.

Parameters

csapatnev	az uj csapatnév.
letszam	az új létszám.

Definition at line 10 of file kosar.cpp.

Here is the call graph for this function:



4.6.3 Member Function Documentation

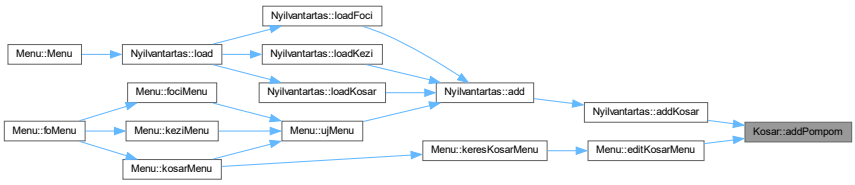
4.6.3.1 addPompom() [1/2]

```
void Kosar::addPompom ( )
```

Egy darab pompomlányt ad hozzá a csapathoz.

Definition at line 15 of file kosar.cpp.

Here is the caller graph for this function:



4.6.3.2 addPompom() [2/2]

```
void Kosar::addPompom (
            const int n )
```

Az adatoknak megfelelő mennyiségű pompomlányt ad a csapathoz.

Parameters

darabSzam	ennyi pompomlányt ad a csapathoz.

Definition at line 17 of file kosar.cpp.

4.6.3.3 getPomPomDb()

```
const int Kosar::getPomPomDb ( ) const
```

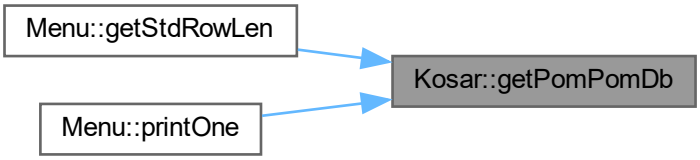
Visszaadja a pompomlányok számát.

Returns

A pompomlányok száma.

Definition at line 19 of file kosar.cpp.

Here is the caller graph for this function:



The documentation for this class was generated from the following files:

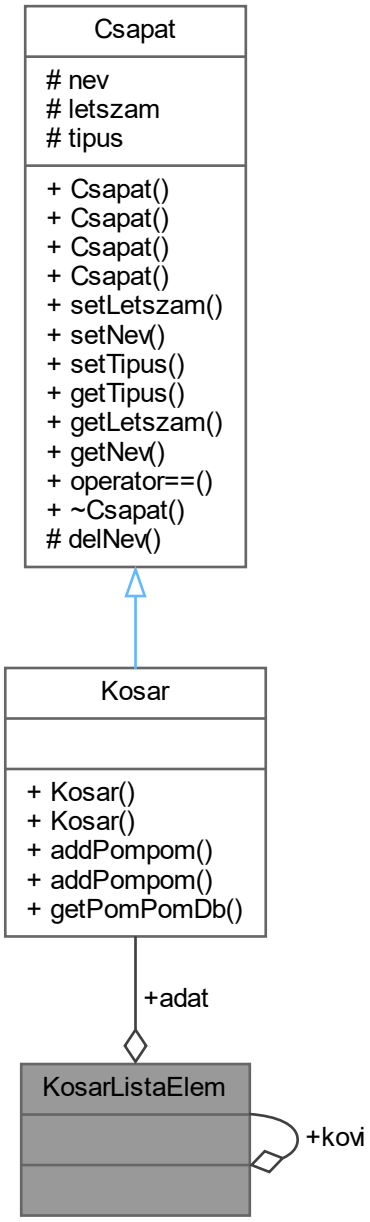
- kosar.h
- kosar.cpp

4.7 KosarListaElem Struct Reference

A Kosar osztály láncolt listázásához szolgáló struktúra. Ez egy darab listaelem.

```
#include <nyilvantartas.h>
```

Collaboration diagram for KosarListaElem:



Public Attributes

- Kosar adat

  A listaelem adata, ami egy Kosar osztály.
- KosarListaElem ∗ kovi

  A következő listaelemre mutató pointer.

4.7.1 Detailed Description

A Kosar osztály láncolt listázásához szolgáló struktúra. Ez egy darab listaelem.

Definition at line 20 of file nyilvantartas.h.

4.7.2 Member Data Documentation

4.7.2.1 adat

```
Kosar KosarListaElem::adat
```

A listaelem adata, ami egy Kosar osztály.

Definition at line 22 of file nyilvantartas.h.

4.7.2.2 kovi

```
KosarListaElem* KosarListaElem::kovi
```

A következő listaelemre mutató pointer.

Definition at line 25 of file nyilvantartas.h.

The documentation for this struct was generated from the following file:

- nyilvantartas.h

4.8 Menu Class Reference

A futó programot irányító menürendszer objektuma.

```
#include <menu.h>
```

Collaboration diagram for Menu:

Public Member Functions

- Menu ()

  Default konstruktor, betölti a nyilvántartás adatait fileból.
- ~Menu ()

  Destruktor, lementi a nyilvántartás adatait fileokba.
- Nyilvantartas getNyilvantartas () const

  Getter. Visszaadja az egész nyilvántartás osztályt (Debug célokra főképp, mivel nincs értelme a classal kommunikálni kívülről.)
- void printAll () const

  Kilistázza megformázva az összes adatot.
- void printKezi () const

  Kilistázza megformázva a Kézilabda (Kezi) csapatokat.
- void printKosar () const

  Kilistázza megformázva a Kárlabda (Kosar) csapatokat.
- void printFoci () const

  Kilistázza megformázva a Focilabda (Foci) csapatokat.
- void printOne (KeziListaElem ∗) const

  Egy listaelemet ír ki megformázva.
- void printOne (KosarListaElem ∗) const

  Egy listaelemet ír ki megformázva.
- void printOne (FociListaElem ∗) const

  Egy listaelemet ír ki megformázva.
- int maxStdRowLen () const

  Kiszámolja a leghoszabb sor hosszát a nyilvántartásban. Erre a TAB-ok és a kinézet miatt van szügség.
- int maxStdRowLen (Tipus) const

  Kiszámolja, hogy a nyílvántartás típus szerinti láncolt listájában mennyi a leghoszabb sor. Erre a TAB-ok és a kinézet miatt van szüggség.
- int getStdRowLen (KeziListaElem ∗) const

  Egy adott listaelem sorának hosszát adja vissza. Design felhasználási céllal.
- int getStdRowLen (KosarListaElem ∗) const

  Egy adott listaelem sorának hosszát adja vissza. Design felhasználási céllal.
- int getStdRowLen (FociListaElem ∗) const

  Egy adott listaelem sorának hosszát adja vissza. Design felhasználási céllal.
- void foMenu ()

  Főmenü. Innen indul minden. Ez tujajdonképpen az entrypoint, ahonnan a class.
- void keziMenu ()

  A kézilabda csapatokkal foglalkozó almenü. Innen lehet kézi sprecifikus dolgokat csinálni.
- void kosarMenu ()

  A kosárlabda csapatokkal foglalkozó almenü. Innen lehet kosár sprecifikus dolgokat csinálni.
- void fociMenu ()

  A focicsapatokkal foglalkozó almenü. Innen lehet foci sprecifikus dolgokat csinálni.
- void keresKeziMenu ()

  Egy kézilabda csapat keresési menüje.
- void keresKosarMenu ()

  Egy kosárlabda csapat keresési menüje.
- void keresFociMenu ()

  Egy focilabda csapat keresési menüje.
- void editKeziMenu (KeziListaElem ∗)

  A láncolt lista egy elemét módosító almenü.
- void editKosarMenu (KosarListaElem ∗)

A láncolt lista egy elemét módosító almenü.

- void editFociMenu (FociListaElem ∗)

  A láncolt lista egy elemét módosító almenü.

- void ujMenu (Tipus)

  Egy új tipus típusú csapat lérehozására létező almenü.

4.8.1 Detailed Description

A futó programot irányító menürendszer objektuma.

Definition at line 8 of file menu.h.

4.8.2 Constructor & Destructor Documentation

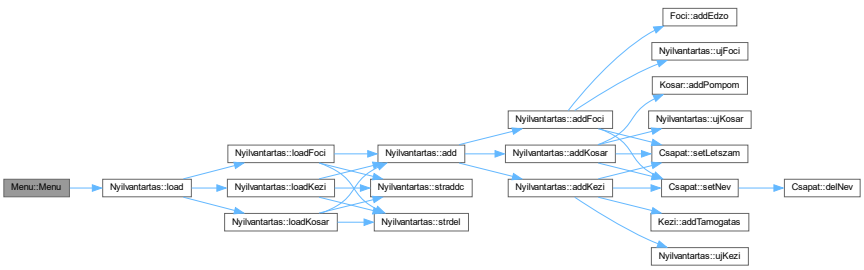
4.8.2.1 Menu()

```
Menu::Menu ( )
```

Default konstruktor, betölti a nyilvántartás adatait fileból.

Definition at line 16 of file menu.cpp.

Here is the call graph for this function:



4.8.2.2 ~Menu()

```
Menu::~Menu ( )
```

Destruktor, lementi a nyilvántartás adatait fileokba.

Definition at line 21 of file menu.cpp.

Here is the call graph for this function:

4.8.3 Member Function Documentation

4.8.3.1 editFociMenu()

```
void Menu::editFociMenu (
            FociListaElem * p )
```

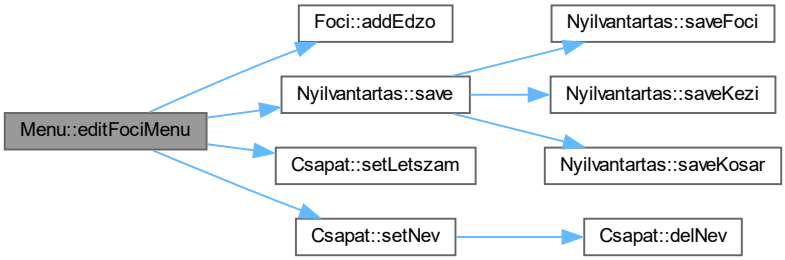
A láncolt lista egy elemét módosító almenü.

Parameters

listaelem	Ennek módosításában segít a menü.

Definition at line 640 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

4.8.3.2 editKeziMenu()

```
void Menu::editKeziMenu (
            KeziListaElem * p )
```

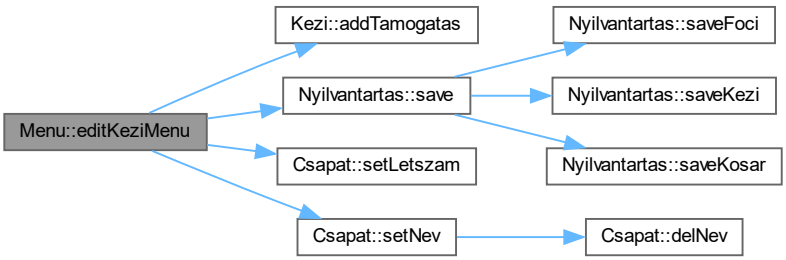
A láncolt lista egy elemét módosító almenü.

Parameters

listaelem	Ennek módosításában segít a menü.

Definition at line 512 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

4.8.3.3 editKosarMenu()

```
void Menu::editKosarMenu (
            KosarListaElem * p )
```

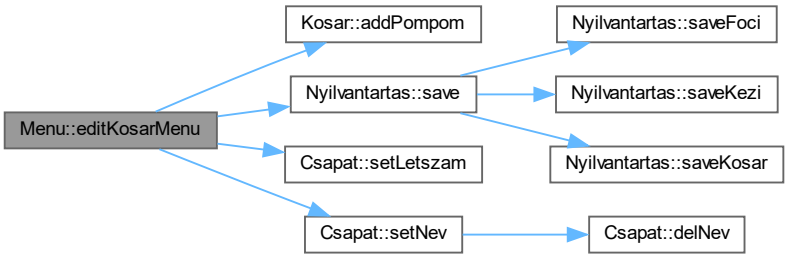
A láncolt lista egy elemét módosító almenü.

Parameters

listaelem	Ennek módosításában segít a menü.

Definition at line 576 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

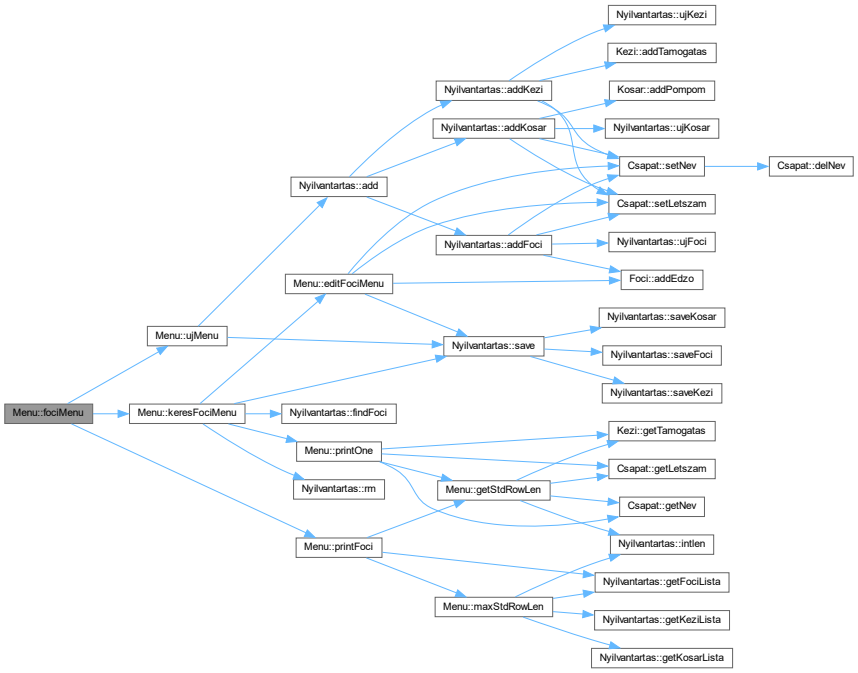
4.8.3.4 fociMenu()

```
void Menu::fociMenu ( )
```

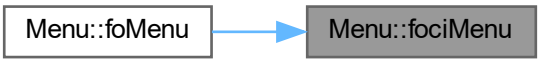
A focicsapatokkal foglalkozó almenü. Innen lehet foci sprecifikus dolgokat csinálni.

Definition at line 308 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.3.5 foMenu()

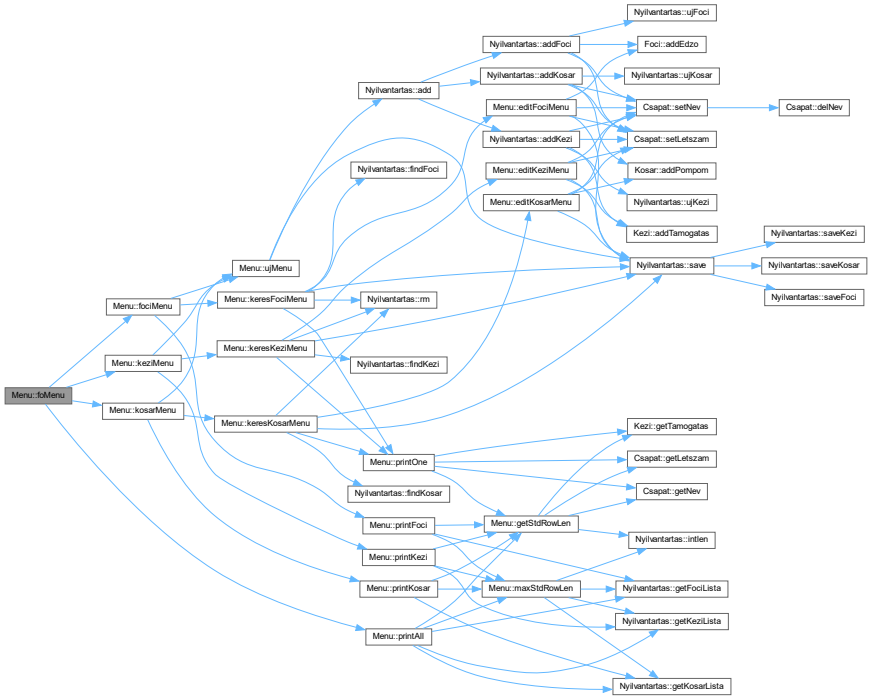
```
void Menu::foMenu ( )
```

Főmenü. Innen indul minden. Ez tujajdonképpen az entrypoint, ahonnan a class.

átveszi az irányítást, és automata menürendszerként üzemel.

Definition at line 751 of file menu.cpp.

Here is the call graph for this function:



4.8.3.6 getNyilvantartas()

```
Nyilvantartas Menu::getNyilvantartas ( ) const  [inline]
```

Getter. Visszaadja az egész nyilvántartás osztályt (Debug célokra főképp, mivel nincs értelme a classal kommunikálni kívülről.)

Returns

Nyilvántartás adatbázis.

Definition at line 22 of file menu.h.

4.8.3.7 getStdRowLen() [1/3]

```
int Menu::getStdRowLen (
            FociListaElem * l ) const
```

Egy adott listaelem sorának hosszát adja vissza. Design felhasználási céllal.

Parameters

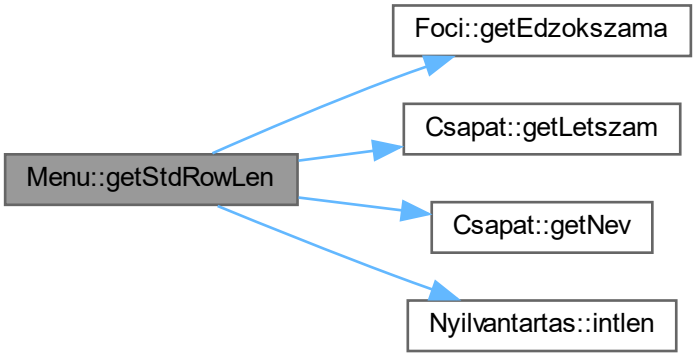
listaelem	Ennek a sorhosszára vagyunk kíváncsiak.

Returns

A listaelem sorának hossza.

Definition at line 92 of file menu.cpp.

Here is the call graph for this function:



4.8.3.8 getStdRowLen() [2/3]

```
int Menu::getStdRowLen (
            KeziListaElem * l ) const
```

Egy adott listaelem sorának hosszát adja vissza. Design felhasználási céllal.

Parameters

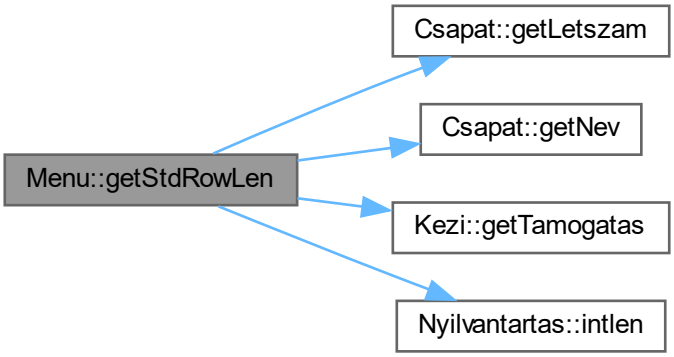
listaelem	Ennek a sorhosszára vagyunk kíváncsiak.

Returns

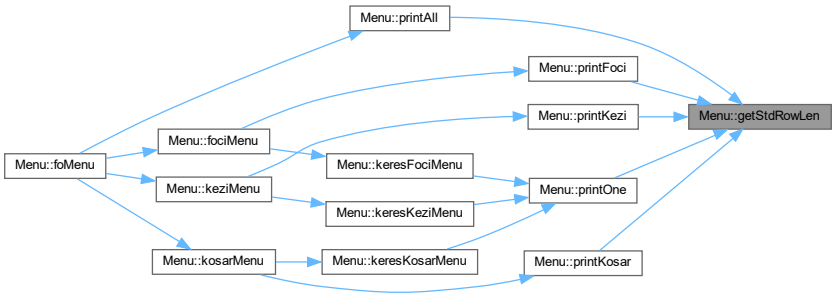
A listaelem sorának hossza.

Definition at line 76 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.3.9 getStdRowLen() [3/3]

```
int Menu::getStdRowLen (
            KosarListaElem * l ) const
```

Egy adott listaelem sorának hosszát adja vissza. Design felhasználási céllal.

Parameters

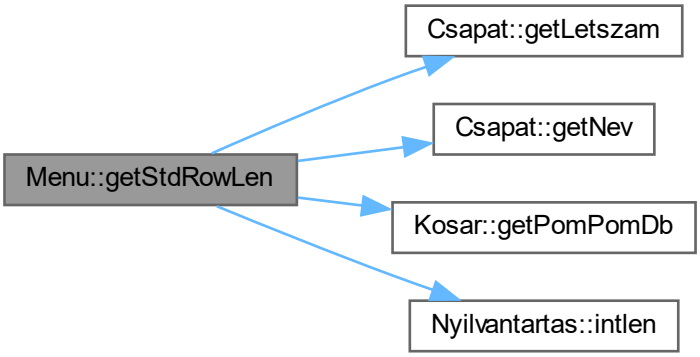
listaelem	Ennek a sorhosszára vagyunk kíváncsiak.

Returns

A listaelem sorának hossza.

Definition at line 84 of file menu.cpp.

Here is the call graph for this function:



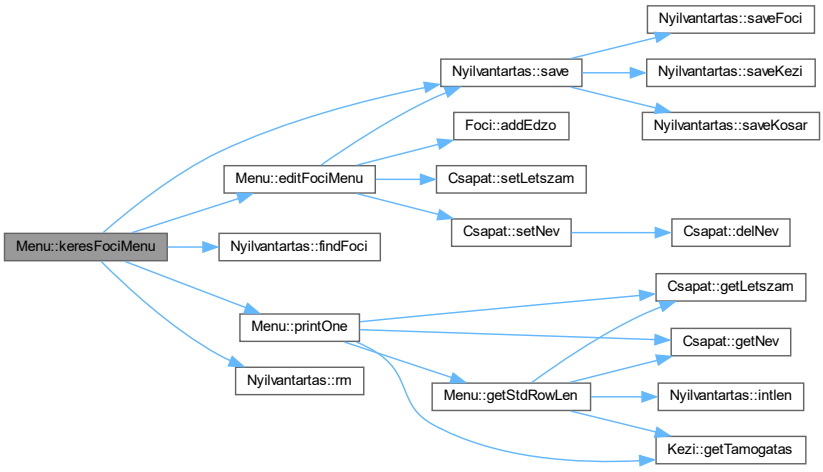
4.8.3.10 keresFociMenu()

```
void Menu::keresFociMenu ( )
```

Egy focilabda csapat keresési menüje.

Definition at line 456 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

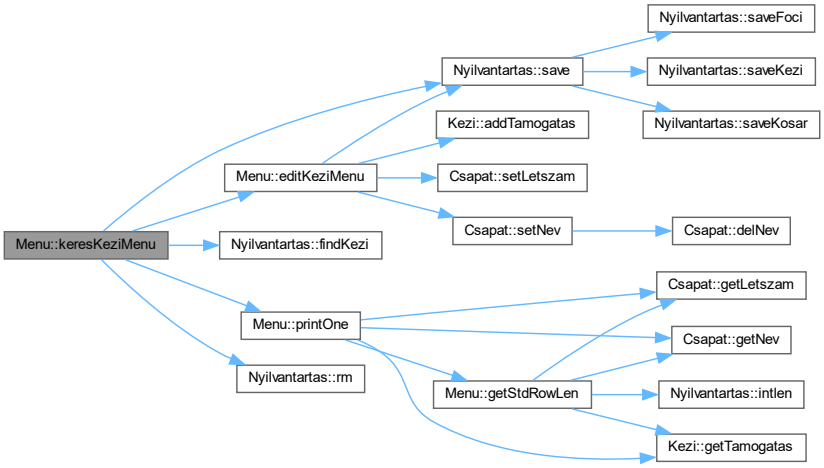
4.8.3.11 keresKeziMenu()

```
void Menu::keresKeziMenu ( )
```

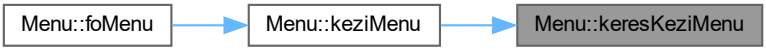
Egy kézilabda csapat keresési menüje.

Definition at line 344 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



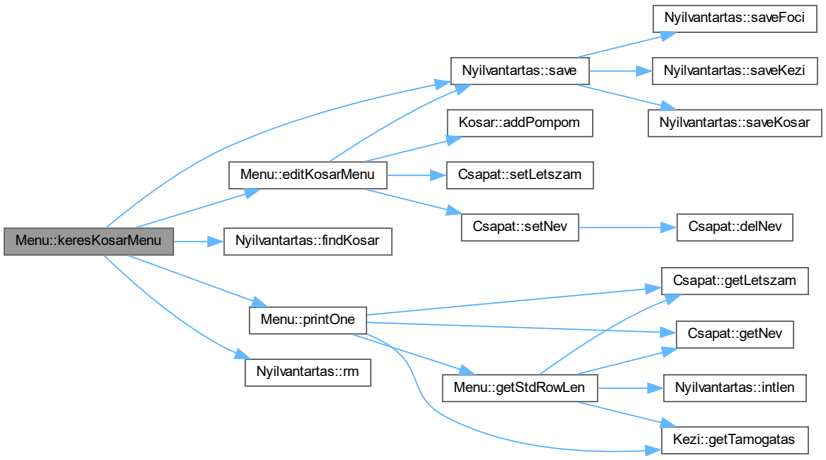
4.8.3.12 keresKosarMenu()

```
void Menu::keresKosarMenu ( )
```

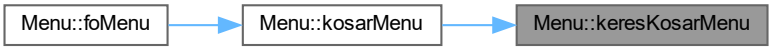
Egy kosárlabda csapat keresési menüje.

Definition at line 400 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



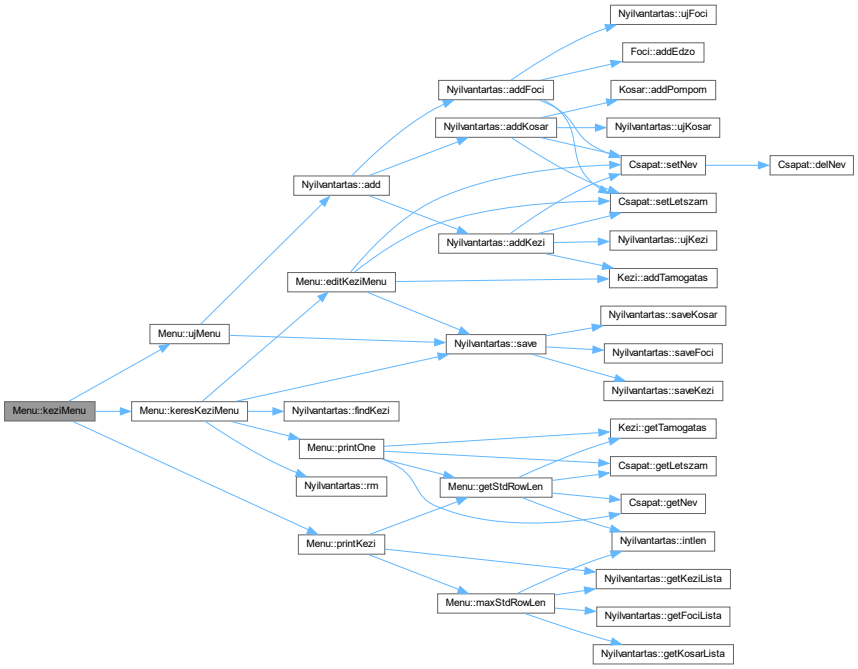
4.8.3.13 keziMenu()

```
void Menu::keziMenu ( )
```

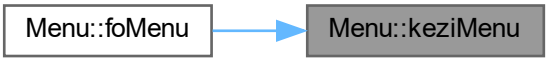
A kézilabda csapatokkal foglalkozó almenü. Innen lehet kézi sprecifikus dolgokat csinálni.

Definition at line 236 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



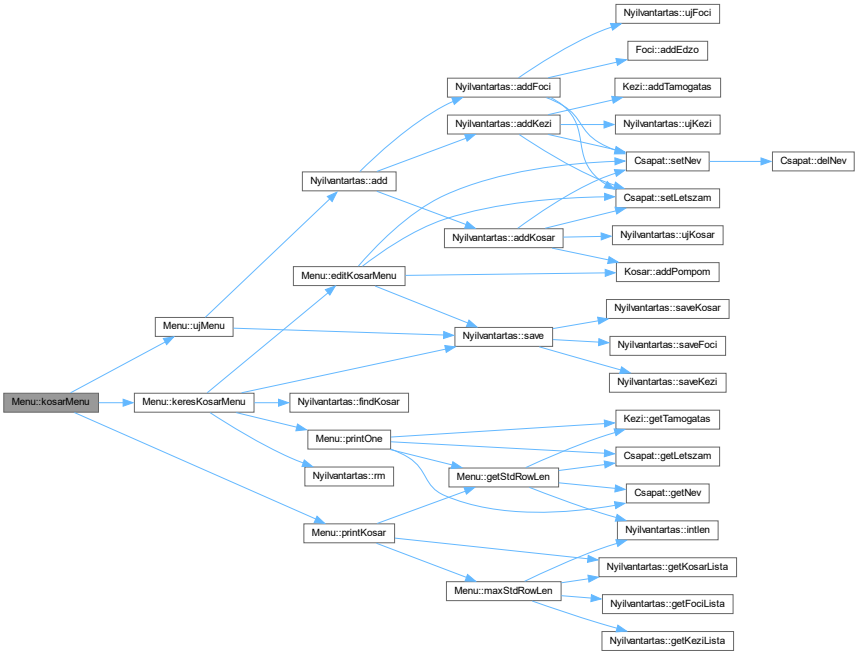
4.8.3.14 kosarMenu()

```
void Menu::kosarMenu ( )
```

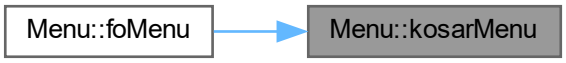
A kosárlabda csapatokkal foglalkozó almenü. Innen lehet kosár sprecifikus dolgokat csinálni.

Definition at line 272 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.3.15 maxStdRowLen() [1/2]

```
int Menu::maxStdRowLen ( ) const
```

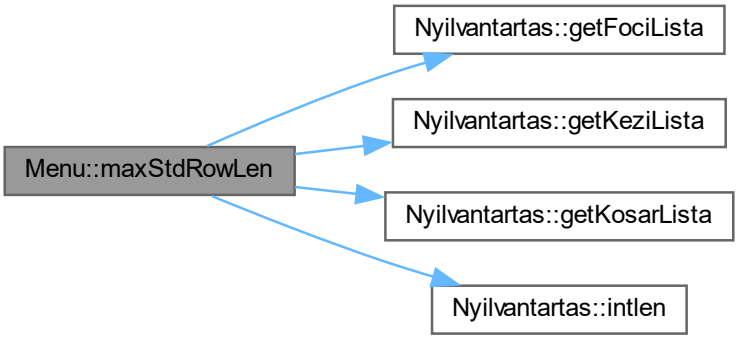
Kiszámolja a leghoszabb sor hosszát a nyilvántartásban. Erre a TAB-ok és a kinézet miatt van szügség.

Returns

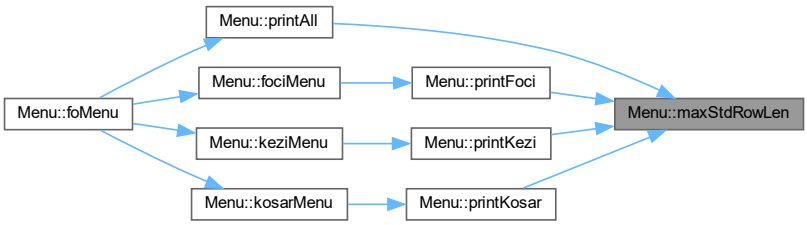
A leghoszabb sor hossza az adatbáisban.

Definition at line 26 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.3.16 maxStdRowLen() [2/2]

```
int Menu::maxStdRowLen (
            Tipus t ) const
```

Kiszámolja, hogy a nyílvántartás típus szerinti láncolt listájában mennyi a leghoszabb sor. Erre a TAB-ok és a kinézet miatt van szüggség.

Parameters

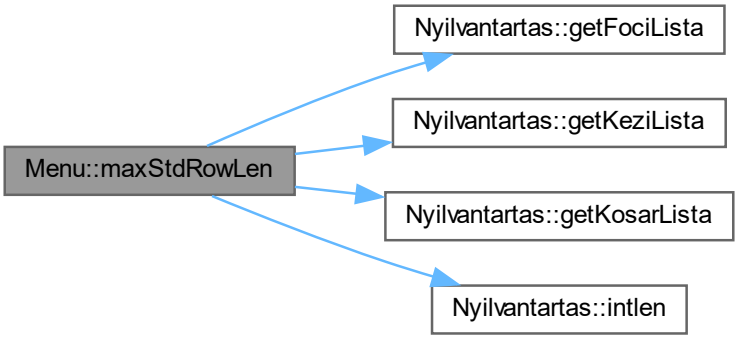
tipus	A láncolt lista típusa, hogy melyikben keresse a leghoszabb sort.

Returns

A maximális sor hossz.

Definition at line 49 of file menu.cpp.

Here is the call graph for this function:



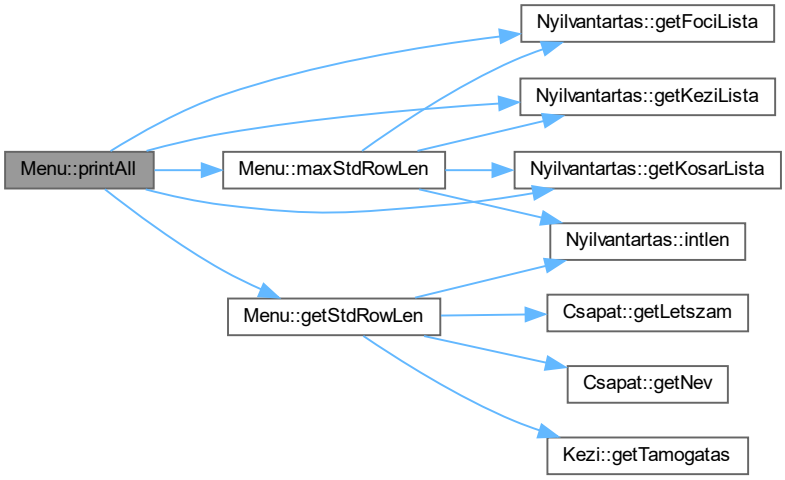
4.8.3.17 printAll()

```
void Menu::printAll ( ) const
```

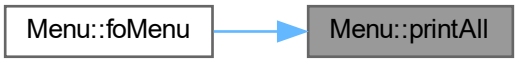
Kilistázza megformázva az összes adatot.

Definition at line 100 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



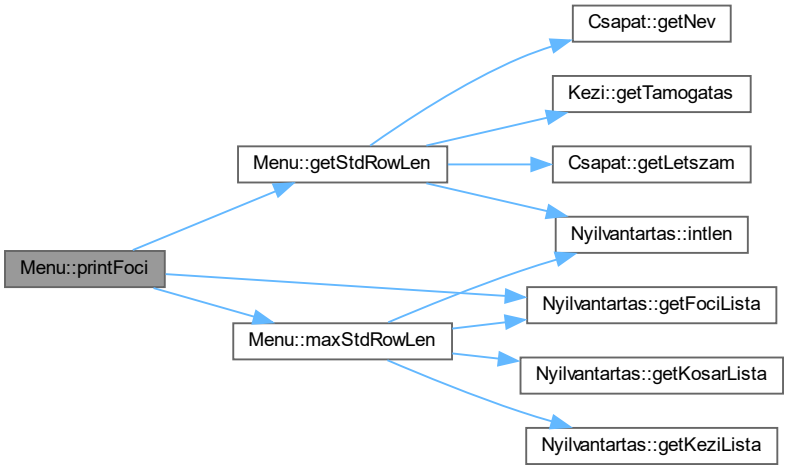
4.8.3.18 printFoci()

```
void Menu::printFoci ( ) const
```

Kilistázza megformázva a Focilabda (Foci) csapatokat.

Definition at line 184 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

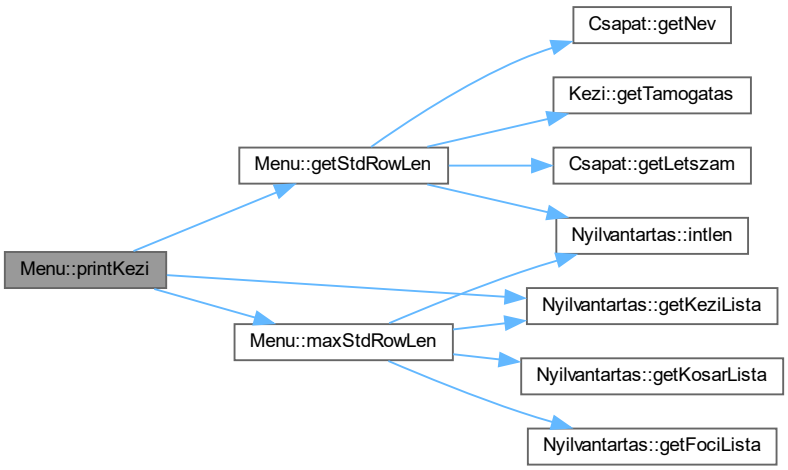
4.8.3.19 printKezi()

```
void Menu::printKezi ( ) const
```

Kilistázza megformázva a Kézilabda (Kezi) csapatokat.

Definition at line 152 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

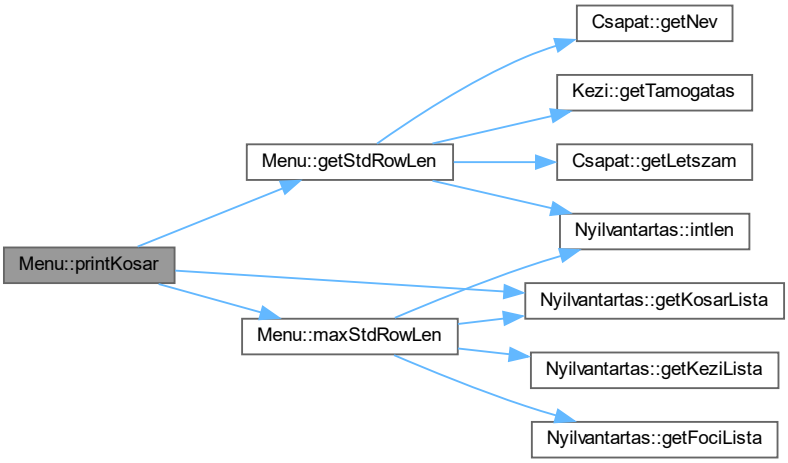
4.8.3.20 printKosar()

```
void Menu::printKosar ( ) const
```

Kilistázza megformázva a Kárlabda (Kosar) csapatokat.

Definition at line 168 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

4.8.3.21 printOne() [1/3]

```
void Menu::printOne (
            FociListaElem * p ) const
```

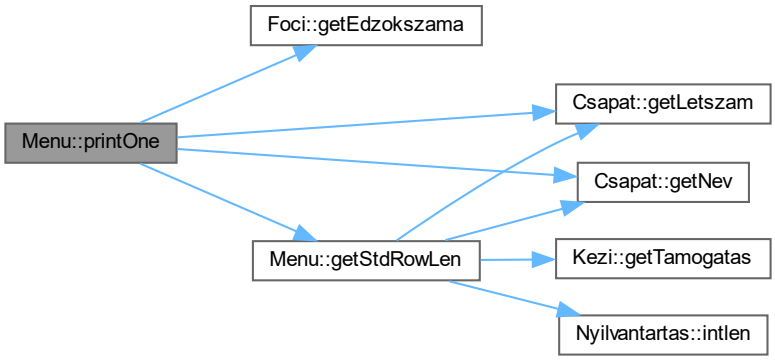
Egy listaelemet ír ki megformázva.

Parameters

listaelem	A lista láncszemére mutató pointer.

Definition at line 224 of file menu.cpp.

Here is the call graph for this function:



4.8.3.22 printOne() [2/3]

```
void Menu::printOne (
            KeziListaElem * p ) const
```

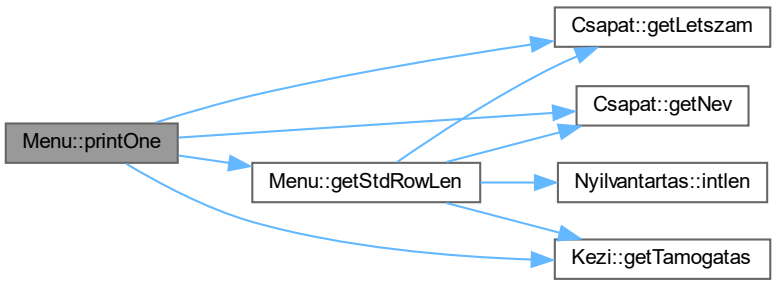
Egy listaelemet ír ki megformázva.

Parameters

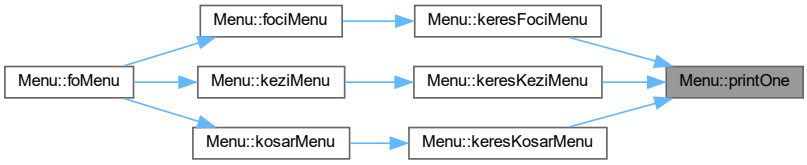
listaelem	A lista láncszemére mutató pointer.

Definition at line 200 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.3.23 printOne() [3/3]

```
void Menu::printOne (
            KosarListaElem * p ) const
```

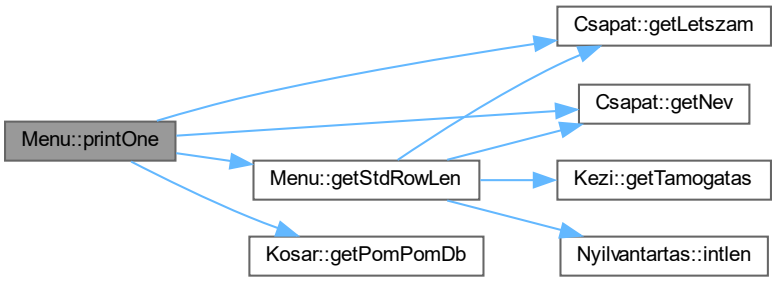
Egy listaelemet ír ki megformázva.

Parameters

listaelem	A lista láncszemére mutató pointer.

Definition at line 212 of file menu.cpp.

Here is the call graph for this function:



4.8.3.24 ujMenu()

```
void Menu::ujMenu (
            Tipus T )
```

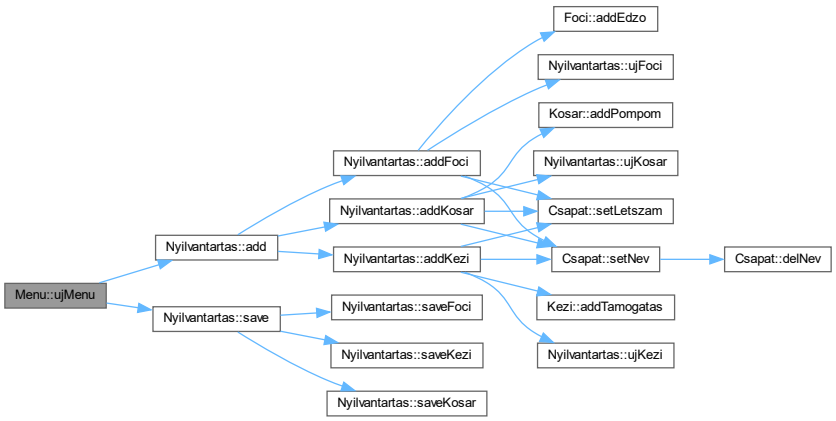
Egy új tipus típusú csapat lérehozására létező almenü.

Parameters

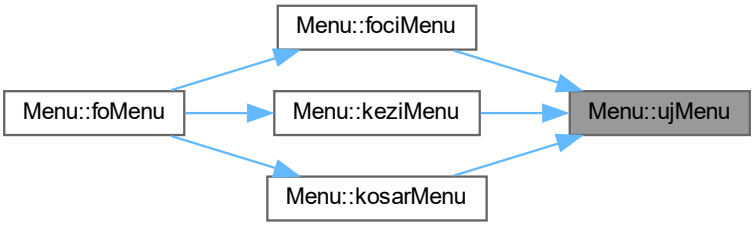
tipus	Az új csapat típusa.

Definition at line 704 of file menu.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following files:

- menu.h
- menu.cpp

4.9 Nyilvantartas Class Reference

A nyilvántartás osztály. Ez tárolja a csapatokat (Kosar, Foci, Kezi) láncolt listákban.

`#include <nyilvantartas.h>`

Collaboration diagram for Nyilvantartas:

Public Member Functions

- Nyilvantartas ()

  A default konstruktor, amely 'nem csinál semmit'. Magyarul inicializála a.
- ~Nyilvantartas ()

  A destruktor, felszabadítja a láncolt listákat a segédfüggvények segítségével.
- KeziListaElem * ujKezi ()

  Létrehoz egy új kézilabda láncolt lista elemet, beláncolja a listába, majd.
- KosarListaElem * ujKosar ()

  Létrehoz egy új kosárlabda láncolt lista elemet, beláncolja a listába, majd.
- FociListaElem * ujFoci ()

  Létrehoz egy új kosárlabda láncolt lista elemet, beláncolja a listába, majd.
- void addKezi (const char *, const int, const int)

  Beleláncol a listába a paramétereknek megfelelő Kézilabda csapatot.
- void addKosar (const char *, const int, const int)

  Beleláncol a listába a paramétereknek megfelelő Kosárlabda csapatot.
- void addFoci (const char *, const int, const int)

  Beleláncol a listába a paramétereknek megfelelő Focilabda csapatot.
- void add (const Tipus, const char *, const int, const int)

  Beleláncol a listába a paramétereknek megfelelő, Típustól függő csapatot csapatot.
- KeziListaElem * findKezi (const char *) const

  A paraméternek megfelelő nevű csapatra mutató pointert ad vissza. Kikeresi a láncolt listából.
- KosarListaElem * findKosar (const char *) const

  A paraméternek megfelelő nevű csapatra mutató pointert ad vissza. Kikeresi a láncolt listából.

- FociListaElem ∗ findFoci (const char ∗) const

  A paraméternek megfelelő nevű csapatra mutató pointert ad vissza. Kikeresi a láncolt listából.
- void rm (KeziListaElem ∗&)

  Kitörli a láncolt listából a paraméterben megadott listaelemet.
- void rm (KosarListaElem ∗&)

  Kitörli a láncolt listából a paraméterben megadott listaelemet.
- void rm (FociListaElem ∗&)

  Kitörli a láncolt listából a paraméterben megadott listaelemet.
- KeziListaElem ∗ getKeziLista () const

  Visszaadja a Kézilabda csapatok láncolt listáját.
- KosarListaElem ∗ getKosarLista () const

  Visszaadja a Kosárlabda csapatok láncolt listáját.
- FociListaElem ∗ getFociLista () const

  Visszaadja a Focilabda csapatok láncolt listáját.
- bool loadKezi ()

  Betölti a kezi.txt fileból a kézilabda csapat adatait a keziCS listába.
- bool loadKosar ()

  Betölti a kosar.txt fileból a kosárlabda csapat adatait a kosarCS listába.
- bool loadFoci ()

  Betölti a foci.txt fileból a kosárlabda csapat adatait a fociCS listába.
- void saveKezi () const

  Elmenti a kezi.txt fileba a keziCS adatait.
- void saveKosar () const

  Elmenti a kosar.txt fileba a kosarCS adatait.
- void saveFoci () const

  Elmenti a foci.txt fileba a fociCS adatait.
- bool load ()

  Betölti az összes fileból (kezi.txt, kosar.txt, foci.txt) az adatokat a.
- void save () const

  Elmenti az összes listát (keziCS, kosarCS, fociCS) a megfelelő fileokba.

Static Public Member Functions

- static int intlen (const long long int)

  Kiszamolja egy szám legnagyobb helyiértékét (tehát, milyen hosszú a szám). Statikus függvény.
- static void straddc (char ∗&, const char)

  Statikus. Hozzáfűz egy karakterpointerhez egy betűt. (VIGYÁZAT UTÁNNA DELETE[]-ELNI KELL).
- static void strdel (char ∗&)

  Felszabadít és nullptr-é tesz egy karakterpointert. Statikus.

4.9.1 Detailed Description

A nyilvántartás osztály. Ez tárolja a csapatokat (Kosar, Foci, Kezi) láncolt listákban.

Képes ezeket fileokból beolvasni, lementeni. Hozzáadni csapatokat, törölni csapatokat.

A program futása során ebből több mint egyet létrehozni nem kell (nem értelmes).

Definition at line 40 of file nyilvantartas.h.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 Nyilvantartas()

```
Nyilvantartas::Nyilvantartas ( )  [inline]
```

A default konstruktor, amely 'nem csinál semmit'. Magyarul inicializála a.

láncolt listákat. (mindegyiket nullptr-re rakja).

Definition at line 107 of file nyilvantartas.h.

4.9.2.2 ~Nyilvantartas()

```
Nyilvantartas::~Nyilvantartas ( )
```

A destruktor, felszabadítja a láncolt listákat a segédfüggvények segítségével.

Definition at line 16 of file nyilvantartas.cpp.

4.9.3 Member Function Documentation

4.9.3.1 add()

```
void Nyilvantartas::add (
            const Tipus T,
            const char * csnev,
            const int letszam,
            const int vari )
```

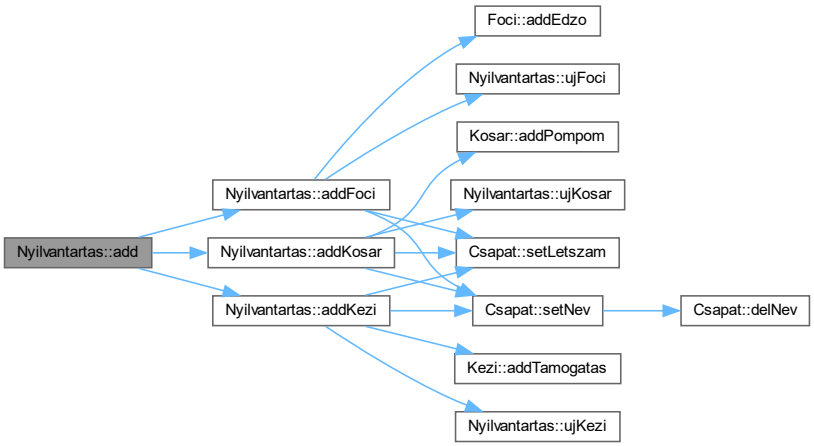
Beleláncol a listába a paramétereknek megfelelő, Típustól függő csapatot csapatot.

Parameters

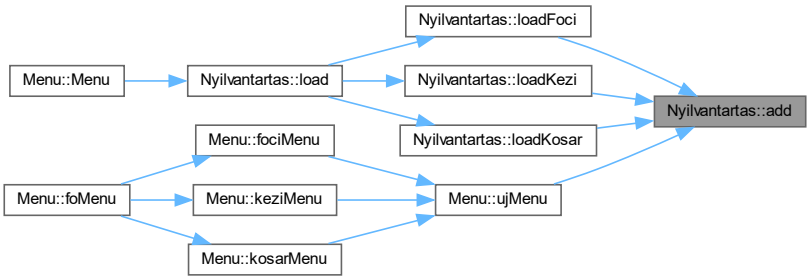
csapat_tipus	Megadja a csapat típusát a Tipus enum segítségével.
csapatnev	a csapat neve.
letszam	a csapat létszáma.
csapat_speicfikus_szam	a csapatokra külön vonatkozó specifikus szám (támogatás, pompomlányok, edzők).

Definition at line 85 of file nyilvantartas.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.3.2 addFoci()

```
void Nyilvantartas::addFoci (
            const char * csnev,
            const int letszam,
            const int edzok )
```

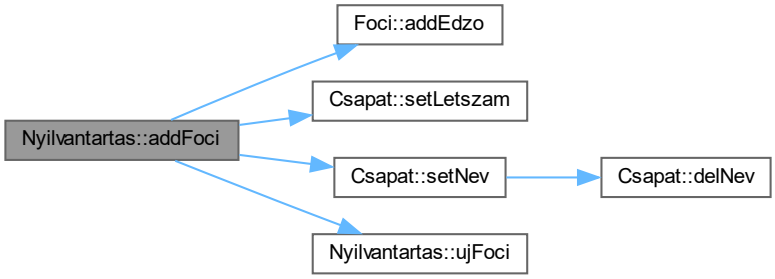
Beleláncol a listába a paramétereknek megfelelő Focilabda csapatot.

Parameters

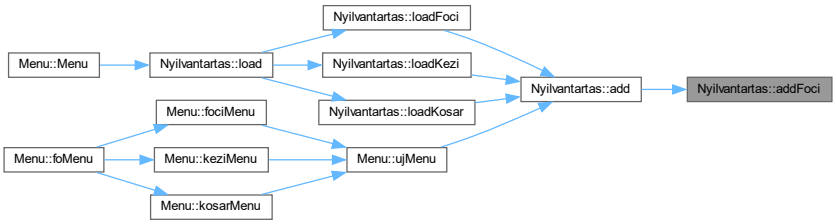
csapatnev	a csapat neve.
letszam	a csapat létszáma.
edzok	a csapat edzőinek száma.

Definition at line 78 of file nyilvantartas.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.3.3 addKezi()

```
void Nyilvantartas::addKezi (
            const char * csnev,
            const int letszam,
            const int tamogatas )
```

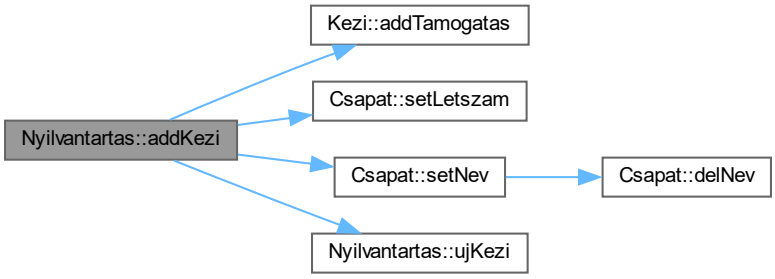
Beleláncol a listába a paramétereknek megfelelő Kézilabda csapatot.

Parameters

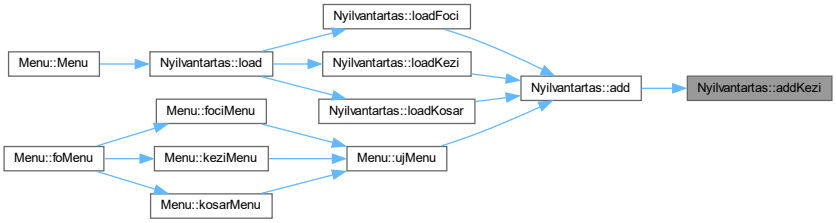
csapatnev	a csapat neve.
letszam	a csapat létszáma.
tamogatas	a csapat támogatásai.

Definition at line 36 of file nyilvantartas.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.3.4 addKosar()

```
void Nyilvantartas::addKosar (
            const char * csnev,
            const int letszam,
            const int pompomn )
```

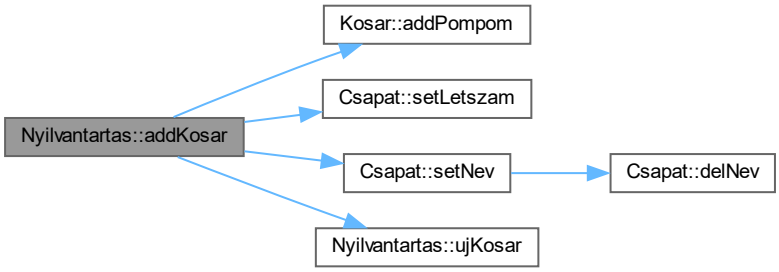
Beleláncol a listába a paramétereknek megfelelő Kosárlabda csapatot.

Parameters

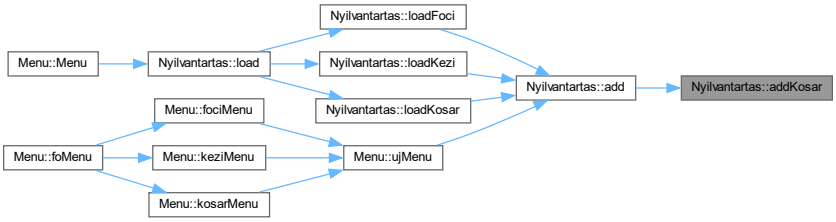
csapatnev	a csapat neve.
letszam	a csapat létszáma.
pompom_lanyok	a csapat pompom lányainak száma.

Definition at line 57 of file nyilvantartas.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.3.5 findFoci()

```
FociListaElem * Nyilvantartas::findFoci (
            const char * csapatnev ) const
```

A paraméternek megfelelő nevű csapatra mutató pointert ad vissza. Kikeresi a láncolt listából.

Parameters

csapat_nev	a keresendő csapat neve.

Returns

A keresendő csapatra mutató pointer, VAGY nullptr ha nem található ilyen csapat.

Definition at line 119 of file nyilvantartas.cpp.

Here is the caller graph for this function:

4.9.3.6 findKezi()

```
KeziListaElem * Nyilvantartas::findKezi (
            const char * csapatnev ) const
```

A paraméternek megfelelő nevű csapatra mutató pointert ad vissza. Kikeresi a láncolt listából.

Parameters

csapat_nev	a keresendő csapat neve.

Returns

A keresendő csapatra mutató pointer, VAGY nullptr ha nem található ilyen csapat.

Definition at line 91 of file nyilvantartas.cpp.

Here is the caller graph for this function:

4.9.3.7 findKosar()

```
KosarListaElem * Nyilvantartas::findKosar (
            const char * csapatnev ) const
```

A paraméternek megfelelő nevű csapatra mutató pointert ad vissza. Kikeresi a láncolt listából.

Parameters

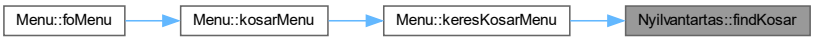
csapat_nev	a keresendő csapat neve.

Returns

A keresendő csapatra mutató pointer, VAGY nullptr ha nem található ilyen csapat.

Definition at line 105 of file nyilvantartas.cpp.

Here is the caller graph for this function:



4.9.3.8 getFociLista()

```
FociListaElem * Nyilvantartas::getFociLista ( ) const
```

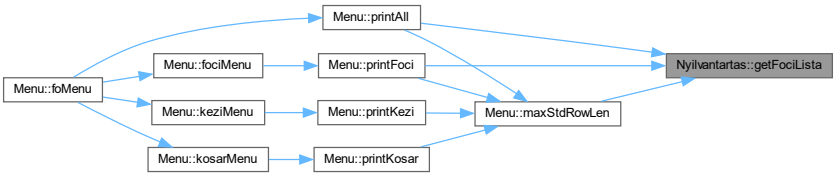
Visszaadja a Focilabda csapatok láncolt listáját.

Returns

Az első listaelemre mutató pointer, vagy nullptr ha üres a lista.

Definition at line 179 of file nyilvantartas.cpp.

Here is the caller graph for this function:



4.9.3.9 getKeziLista()

```
KeziListaElem * Nyilvantartas::getKeziLista ( ) const
```

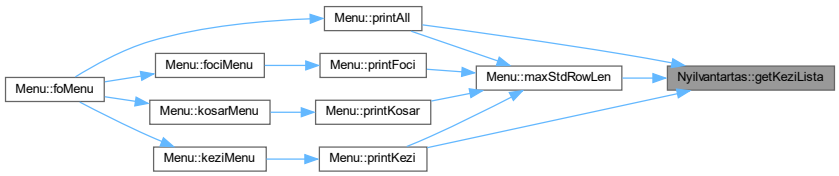
Visszaadja a Kézilabda csapatok láncolt listáját.

Returns

Az első listaelemre mutató pointer, vagy nullptr ha üres a lista.

Definition at line 175 of file nyilvantartas.cpp.

Here is the caller graph for this function:



4.9.3.10 getKosarLista()

```
KosarListaElem * Nyilvantartas::getKosarLista ( ) const
```

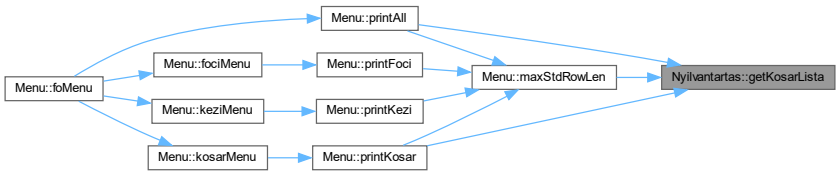
Visszaadja a Kosárlabda csapatok láncolt listáját.

Returns

Az első listaelemre mutató pointer, vagy nullptr ha üres a lista.

Definition at line 177 of file nyilvantartas.cpp.

Here is the caller graph for this function:



4.9.3.11 intlen()

```
int Nyilvantartas::intlen (
            const long long int szam )  [static]
```

Kiszamolja egy szám legnagyobb helyiértékét (tehát, milyen hosszú a szám). Statikus függvény.

Nem működik tökéletesen, de 63 számjegyig működik (egyébként sem reális 64 számjegyű támogatás, vagy pompomlányok szóval most jó lesz...)

Parameters

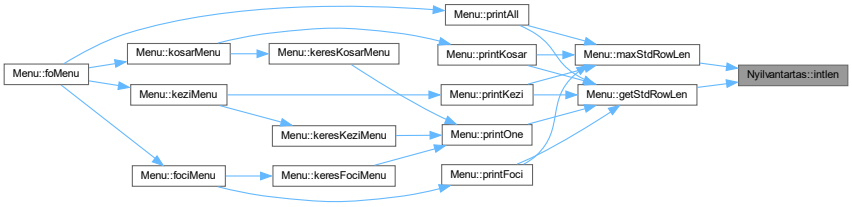
szam	a kiszámolandó szám.

Returns

A szám legnagyobb helyiértéke.

Definition at line 329 of file nyilvantartas.cpp.

Here is the caller graph for this function:



4.9.3.12 load()

```
bool Nyilvantartas::load ( )
```

Betölti az összes fileból (kezi.txt, kosar.txt, foci.txt) az adatokat a.

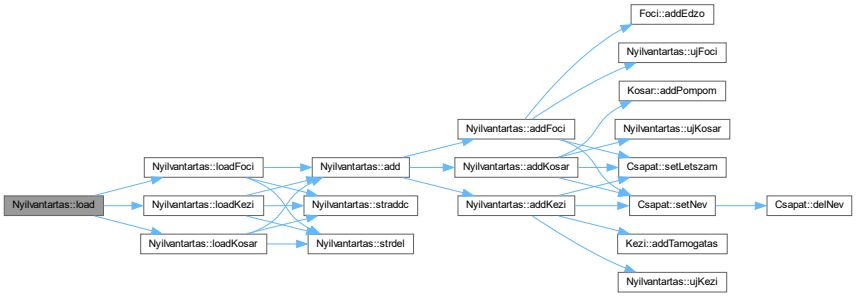
listákba (keziCS, kosarCS, fociCS) a segédfüggvények segítségével.

Returns

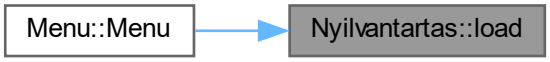
igaz ha sikeres, hamis ha nem sikeres.

Definition at line 322 of file nyilvantartas.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.3.13 loadFoci()

```
bool Nyilvantartas::loadFoci ( )
```

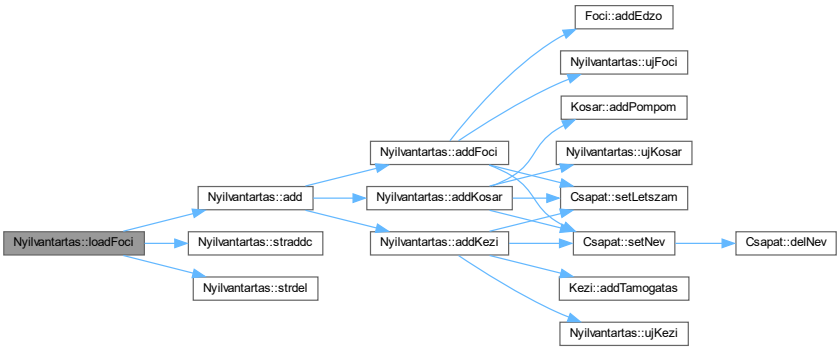
Betölti a foci.txt fileból a kosárlabda csapat adatait a fociCS listába.

Returns

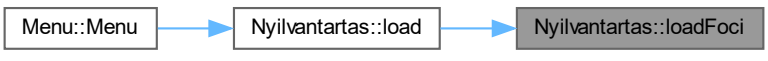
Igaz, ha sikerült betölteni, hamis ha nem.

Definition at line 275 of file nyilvantartas.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.3.14 loadKezi()

```
bool Nyilvantartas::loadKezi ( )
```

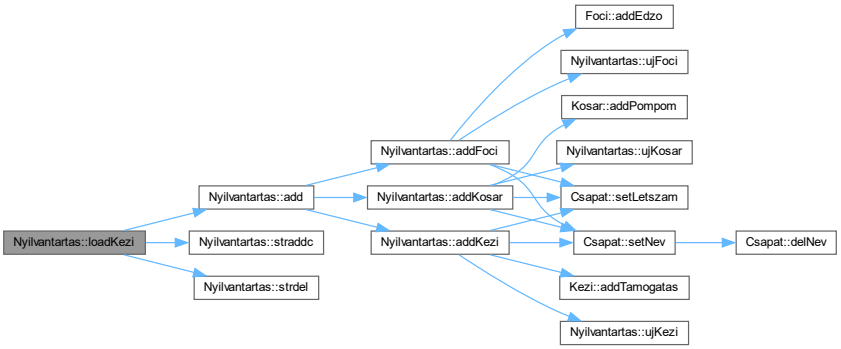
Betölti a kezi.txt fileból a kézilabda csapat adatait a keziCS listába.

Returns

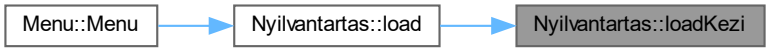
Igaz, ha sikerült betölteni, hamis ha nem.

Definition at line 181 of file nyilvantartas.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.3.15 loadKosar()

```
bool Nyilvantartas::loadKosar ( )
```

Betölti a kosar.txt fileból a kosárlabda csapat adatait a kosarCS listába.

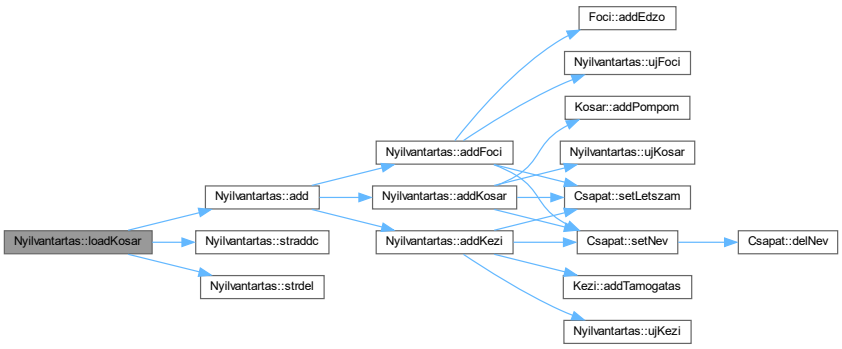
Returns

Igaz, ha sikerült betölteni, hamis ha nem.

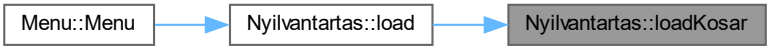
File leírás:

Definition at line 228 of file nyilvantartas.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.3.16 rm() [1/3]

```
void Nyilvantartas::rm (
            FociListaElem *& torlendo )
```

Kitörli a láncolt listából a paraméterben megadott listaelemet.

Parameters

lista_elem	a láncolt listában egy elem-re mutató pointer.

Definition at line 161 of file nyilvantartas.cpp.

4.9.3.17 rm() [2/3]

```
void Nyilvantartas::rm (
            KeziListaElem *& torlendo )
```

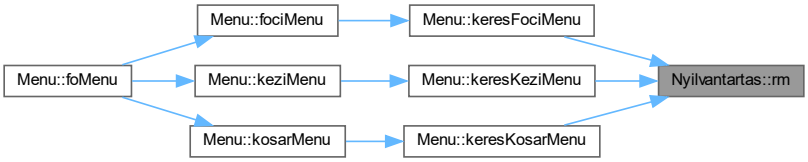
Kitörli a láncolt listából a paraméterben megadott listaelemet.

Parameters

lista_elem	a láncolt listában egy elem-re mutató pointer.

Definition at line 133 of file nyilvantartas.cpp.

Here is the caller graph for this function:



4.9.3.18 rm() [3/3]

```
void Nyilvantartas::rm (
            KosarListaElem *& torlendo )
```

Kitörli a láncolt listából a paraméterben megadott listaelemet.

Parameters

lista_elem	a láncolt listában egy elem-re mutató pointer.

Definition at line 147 of file nyilvantartas.cpp.

4.9.3.19 save()

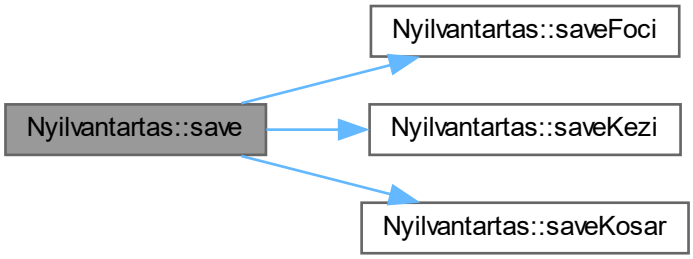
```
void Nyilvantartas::save ( ) const
```

Elmenti az összes listát (keziCS, kosarCS, fociCS) a megfelelő fileokba.

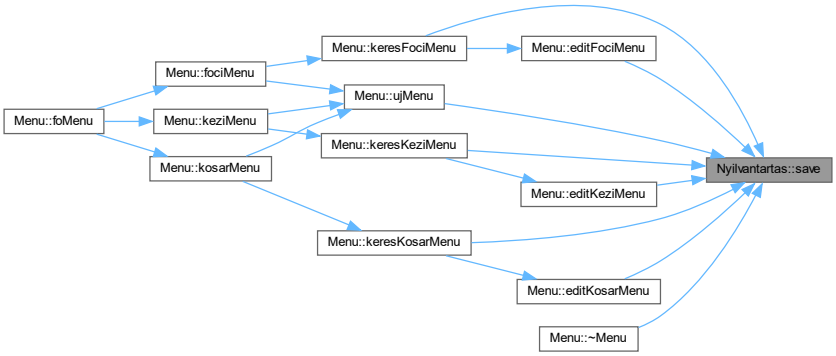
(kezi.txt, kosar.txt, foci.txt) a segédfüggvények segítségével.

Definition at line 393 of file nyilvantartas.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



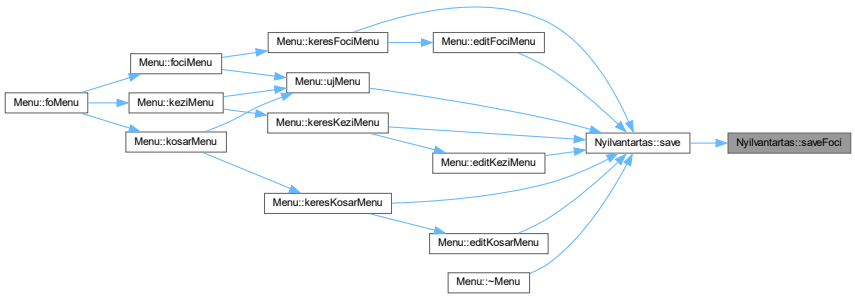
4.9.3.20 saveFoci()

```
void Nyilvantartas::saveFoci ( ) const
```

Elmenti a foci.txt fileba a fociCS adatait.

Definition at line 382 of file nyilvantartas.cpp.

Here is the caller graph for this function:



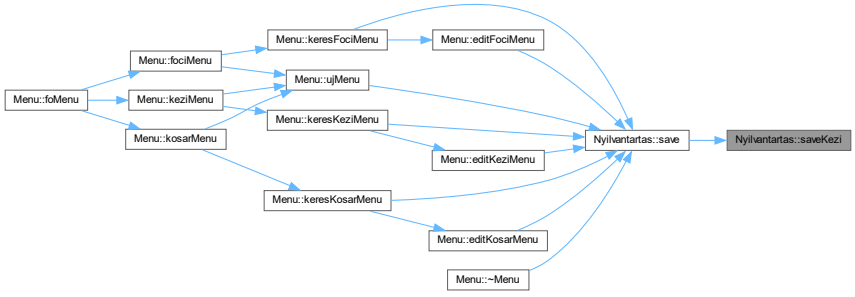
4.9.3.21 saveKezi()

```
void Nyilvantartas::saveKezi ( ) const
```

Elmenti a kezi.txt fileba a keziCS adatait.

Definition at line 360 of file nyilvantartas.cpp.

Here is the caller graph for this function:



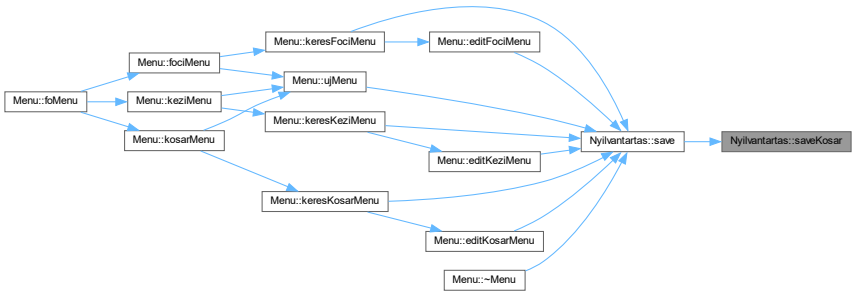
4.9.3.22 saveKosar()

```
void Nyilvantartas::saveKosar ( ) const
```

Elmenti a kosar.txt fileba a kosarCS adatait.

Definition at line 371 of file nyilvantartas.cpp.

Here is the caller graph for this function:



4.9.3.23 straddc()

```
void Nyilvantartas::straddc (
            char *& str,
            const char c )  [static]
```

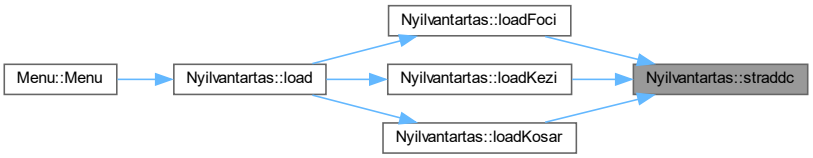
Statikus. Hozzáfűz egy karakterpointerhez egy betűt. (VIGYÁZAT UTÁNNA DELETE[]-ELNI KELL).

Parameters

str	A karakterpointer.
c	A hozzáadandó betű.

Definition at line 338 of file nyilvantartas.cpp.

Here is the caller graph for this function:



4.9.3.24 strdel()

```
void Nyilvantartas::strdel (
            char *& str )  [static]
```

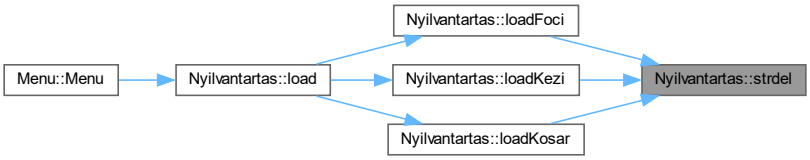
Felszabadít és nullptr-é tesz egy karakterpointert. Statikus.

Parameters

str	A karakertpointer.

Definition at line 355 of file nyilvantartas.cpp.

Here is the caller graph for this function:



4.9.3.25 ujFoci()

```
FociListaElem * Nyilvantartas::ujFoci ( )
```

Létrehoz egy új kosárlabda láncolt lista elemet, beláncolja a listába, majd.

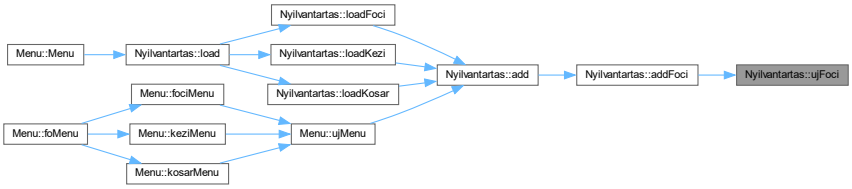
visszaadja az erre mutató pointer (azért, hogy lehessen vele dolgozni.)

Returns

Az újjonnan létrehozott láncolt listaelem pointere.

Definition at line 64 of file nyilvantartas.cpp.

Here is the caller graph for this function:



4.9.3.26 ujKezi()

```
KeziListaElem * Nyilvantartas::ujKezi ( )
```

Létrehoz egy új kézilabda láncolt lista elemet, beláncolja a listába, majd.

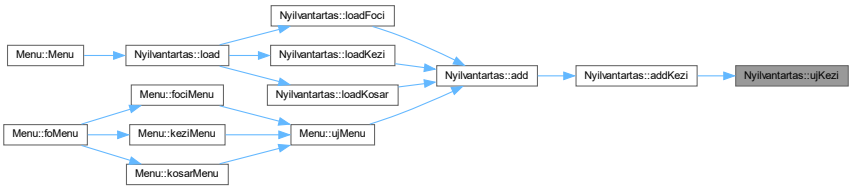
visszaadja az erre mutató pointer (azért, hogy lehessen vele dolgozni.)

Returns

Az újjonnan létrehozott láncolt listaelem pointere.

Definition at line 22 of file nyilvantartas.cpp.

Here is the caller graph for this function:



4.9.3.27 ujKosar()

```
KosarListaElem * Nyilvantartas::ujKosar ( )
```

Létrehoz egy új kosárlabda láncolt lista elemet, beláncolja a listába, majd.

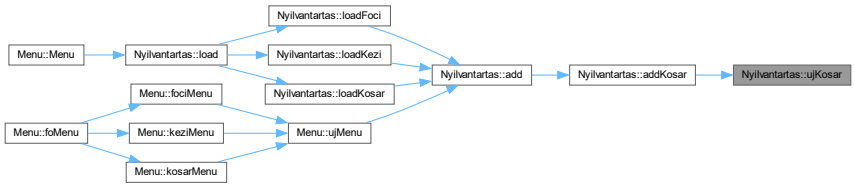
visszaadja az erre mutató pointer (azért, hogy lehessen vele dolgozni.)

Returns

Az újjonnan létrehozott láncolt listaelem pointere.

Definition at line 43 of file nyilvantartas.cpp.

Here is the caller graph for this function:



The documentation for this class was generated from the following files:

- nyilvantartas.h
- nyilvantartas.cpp

Chapter 5

File Documentation

5.1 csapat.cpp

```
00001 #include <cstring>
00002 #include "csapat.h"
00003 #include "memtrace.h"
00004
00005 Csapat::Csapat(const char* p, int n) : letszam(n), tipus(NINCS) {
00006   nev = new char[strlen(p)+1];
00007   strcpy(nev, p);
00008 }
00009
00010 Csapat::Csapat(const char* p) : letszam(0), tipus(NINCS) {
00011   nev = new char[strlen(p)+1];
00012   strcpy(nev, p);
00013 }
00014
00015 Csapat::Csapat(int n) : nev(nullptr), letszam(n), tipus(NINCS) {}
00016
00017 void Csapat::setLetszam(int n) {
00018   letszam = n;
00019 }
00020
00021 void Csapat::setNev(const char* p) {
00022   delNev();
00023   nev = new char[strlen(p)+1];
00024   strcpy(nev, p);
00025 }
00026
00027 void Csapat::setTipus(const Tipus t) {
00028   tipus = t;
00029 }
00030
00031 Tipus Csapat::getTipus() const {
00032   return tipus;
00033 }
00034
00035 int Csapat::getLetszam() const {
00036   return letszam;
00037 }
00038
00039 const char *Csapat::getNev() const {
00040   return nev;
00041 }
00042
00043 bool Csapat::operator==(const char *str){
00044   return (strcmp(nev, str) == 0) ? true : false;
00045 }
00046
00047 Csapat::~Csapat(){
00048   delNev();
00049 }
```

5.2 csapat.h

```
00001 #ifndef CSAPAT_H
00002 #define CSAPAT_H
```

```
00003
00004 #include "memtrace.h"
00005
00007 enum Tipus {
00009   NINCS,
00010
00012   KEZI,
00013
00015   FOCI,
00016
00018   KOSAR
00019 };
00020
00023 class Csapat {
00024   protected:
00026     char *nev;
00027
00029     int letszam;
00030
00032     Tipus tipus;
00033
00035     void delNev() { if (nev != nullptr) { delete[] nev; } };
00036
00037   public:
00039     Csapat() : nev(nullptr), letszam(0), tipus(NINCS) {};
00040
00044     Csapat(const char *, int);
00045
00048     Csapat(const char *);
00049
00052     Csapat(int);
00053
00056     void setLetszam(int);
00057
00060     void setNev(const char *);
00061
00064     void setTipus(const Tipus);
00065
00068     Tipus getTipus() const;
00069
00072     int getLetszam() const;
00073
00076     const char *getNev() const;
00077
00083     bool operator==(const char*);
00084
00086     virtual ~Csapat();
00087
00088 };
00089
00090 #endif
```

5.3 foci.cpp

```
00001 #include "csapat.h"
00002 #include "foci.h"
00003 #include "memtrace.h"
00004
00005 Foci::Foci() : Csapat(0), edzoDB(0) {
00006   setTipus(FOCI);
00007 }
00008
00009 Foci::Foci(const char *p = "", const int n = 0) :
00010   Csapat(p, n), edzoDB(0) {
00011   setTipus(FOCI);
00012 }
00013
00014 void Foci::addEdzo() {edzoDB++;}
00015
00016 void Foci::addEdzo(const int n) {edzoDB = n;}
00017
00018 const int Foci::getEdzokszama() const {return edzoDB;}
00019
```

5.4 foci.h

```
00001 #ifndef FOCI_H
00002 #define FOCI_H
00003
```

```
00004 #include "csapat.h"
00005 #include "memtrace.h"
00006
00008 class Foci : public Csapat {
00009   private:
00011     int edzoDB;
00012
00013   public:
00016     Foci();
00017
00021     Foci(const char *, const int);
00022
00024     void addEdzo();
00025
00028     void addEdzo(const int);
00029
00032     const int getEdzokszama() const;
00033
00034 };
00035
00036 #endif
```

5.5 kezi.cpp

```
00001 #include "csapat.h"
00002 #include "kezi.h"
00003 #include "memtrace.h"
00004
00005 Kezi::Kezi() : Csapat(0), tamogatas(0) {
00006   setTipus(KEZI);
00007 }
00008
00009 Kezi::Kezi(const char* p, const int n) :
00010   Csapat(p, n), tamogatas(0) {
00011   setTipus(KEZI);
00012 }
00013
00014 void Kezi::addTamogatas(const int t) {
00015   tamogatas += t;
00016 }
00017
00018 const int Kezi::getTamogatas() const {
00019   return tamogatas;
00020 }
```

5.6 kezi.h

```
00001 #ifndef KEZI_H
00002 #define KEZI_H
00003
00004 #include "csapat.h"
00005 #include "memtrace.h"
00006
00008 class Kezi : public Csapat {
00009   private:
00011     int tamogatas;
00012
00013   public:
00016     Kezi();
00017
00021     Kezi(const char*, const int);
00022
00025     void addTamogatas(const int);
00026
00029     const int getTamogatas() const;
00030 };
00031
00032 #endif
```

5.7 kosar.cpp

```
00001 #include "csapat.h"
00002 #include "kosar.h"
00003 #include "memtrace.h"
00004
```

```
00005
00006 Kosar::Kosar() : Csapat(0), pompomDB(0) {
00007   setTipus(KOSAR);
00008 }
00009
00010 Kosar::Kosar(const char* p = "", const int n = 0) :
00011   Csapat(p, n), pompomDB(0) {
00012   setTipus(KOSAR);
00013 }
00014
00015 void Kosar::addPompom() {pompomDB++;}
00016
00017 void Kosar::addPompom(const int n) {pompomDB = n;}
00018
00019 const int Kosar::getPomPomDb() const {return pompomDB;}
```

5.8 kosar.h

```
00001 #ifndef KOSAR_H
00002 #define KOSAR_H
00003
00004 #include "csapat.h"
00005 #include "memtrace.h"
00006
00008 class Kosar : public Csapat {
00009   private:
00011     int pompomDB;
00012
00013   public:
00016     Kosar();
00017
00021     Kosar(const char*, const int);
00022
00024     void addPompom();
00025
00028     void addPompom(const int);
00029
00032     const int getPomPomDb() const;
00033
00034 };
00035
00036 #endif
```

5.9 main.cpp

```
00001 #include <iostream>
00002 #include "menu.h"
00003 #include "memtrace.h"
00004
00005 using std::cout;
00006 using std::endl;
00007
00008 void scopeScript(){
00009   Menu m;
00010   m.foMenu();
00011 }
00012
00013 int main() {
00014   scopeScript();
00015   cout « "[-]PROGRAMVEGE" « endl;
00016   return 0;
00017 }
```

5.10 memtrace.cpp

```
00001 /*********************************
00002 Memoriaszivargas-detektor
00003 Keszitette: Peregi Tamas, BME IIT, 2011
00004             petamas@iit.bme.hu
00005 Kanari:     Szeberenyi Imre, 2013.
00006 VS 2012:    Szeberényi Imre, 2015.,
00007 mem_dump:   2016.
00008 meset felszabaditaskor: 2018.
00009 typo:       2019.
00010 *********************************/
```

```
00011
00012 /*definialni kell, ha nem paracssorbol allitjuk be (-DMEMTRACE) */
00013 /*#define MEMTRACE */
00014
00015 #ifdef _MSC_VER
00016     #define _CRT_SECURE_NO_WARNINGS 1
00017 #endif
00018
00019 #include <stdio.h>
00020 #include <stdlib.h>
00021 #include <string.h>
00022 #include <time.h>
00023 #include <ctype.h>
00024
00025 #ifdef MEMTRACE
00026 #define FROM_MEMTRACE_CPP
00027 #include "memtrace.h"
00028
00029 #define FMALLOC 0
00030 #define FCALLOC 1
00031 #define FREALLOC 2
00032 #define FFREE 3
00033 #define FNEW 4
00034 #define FDELETE 5
00035 #define FNEWARR 6
00036 #define FDELETEARR 7
00037 #define COMP(a,d) (((a)<=3 && (d)<=3) || ((d)==(a)+1))
00038 #define PU(p)   ((char*)p+CANARY_LEN)   // mem pointerbol user poi
00039 #define P(pu)   ((char*)pu-CANARY_LEN)  // user pointerbol mem poi
00040 #define XSTR(s) STR(s)
00041 #define STR(s)  #s
00042 /*****************************************************************/
00043 /* Segedfuggvenyek es egyebek */
00044 /*****************************************************************/
00045 START_NAMESPACE
00046     static FILE *fperror;
00047     #ifdef MEMTRACE_TO_MEMORY
00048         static const unsigned int CANARY_LEN = 64;
00049     #else
00050         static const unsigned int CANARY_LEN = 0;
00051     #endif
00052     static const unsigned char canary_byte1 = 'k';
00053     static const unsigned char canary_byte2 = 'K';
00054     static unsigned char random_byte;
00055
00056     typedef enum {FALSE,TRUE} BOOL;
00057
00058     static const char * pretty[] = {"malloc(", "calloc(", "realloc(", "free(",
00059                                     "new", "delete", "new[]", "delete[]"};
00060
00061     static const char * basename(const char * s) {
00062         const char *s1,*s2;
00063         s1 = strrchr(s,'/');
00064         if(s1==NULL) s1 = s; else s1++;
00065         s2 = strrchr(s1, '\\');
00066         if(s2==NULL) s2 = s1; else s2++;
00067         return s2;
00068     }
00069
00070     static char *StrCpy(char ** to, const char * from) {
00071             if(from == NULL) {
00072             *to = NULL;
00073             } else {
00074                     *to = (char*)malloc(strlen(from)+1);
00075                     if(*to) strcpy(*to, from);
00076             }
00077             return *to;
00078     }
00079
00080     static void *canary_malloc(size_t size, unsigned char data) {
00081         char *p = (char *)malloc(size+2*CANARY_LEN);
00082         if (p) {
00083             memset(p, canary_byte1, CANARY_LEN);
00084             memset(p+CANARY_LEN, data, size);
00085             memset(p+CANARY_LEN+size, canary_byte2, CANARY_LEN);
00086         }
00087         return p;
00088     }
00089
00090     static int chk_canary(void *p, size_t size) {
00091         unsigned char *pc = (unsigned char*)p;
00092         unsigned int i;
00093         for (i = 0; i < CANARY_LEN; i++)
00094             if (pc[i] != canary_byte1)
00095                 return -1;
00096         pc += CANARY_LEN+size;
00097         for (i = 0; i < CANARY_LEN; i++)
```

```
00098             if (pc[i] != canary_byte2)
00099                 return 1;
00100         return 0;
00101     }
00102
00103     typedef struct {
00104         int f;  /* allocator func */
00105         int line;
00106         char * par_txt;
00107         char * file;
00108     } call_t;
00109
00110     static call_t pack(int f, const char * par_txt, int line, const char * file) {
00111         call_t ret;
00112         ret.f = f;
00113         ret.line = line;
00114         StrCpy(&ret.par_txt, par_txt);
00115         StrCpy(&ret.file, file);
00116         return ret;
00117     }
00118
00119     static void print_call(const char * msg, call_t call) {
00120         if(msg) fprintf(fperror, "%s", msg);
00121         fprintf(fperror, "%s", pretty[call.f]);
00122         fprintf(fperror, "%s", call.par_txt ? call.par_txt : "?");
00123         if (call.f <= 3) fprintf(fperror, ")");
00124         fprintf(fperror," @ %s:", call.file ? basename(call.file) : "?");
00125         fprintf(fperror,"%d\n",call.line ? call.line : 0);
00126     }
00127
00128     /* memoriateruletet dump */
00129     static void dump_memory(void const *mem, size_t size, size_t can_len, FILE* fp) {
00130         unsigned char const *m=(unsigned char const *) mem;
00131         unsigned int s, o;
00132
00133         if (can_len > 0)
00134             fprintf(fp, "Dump (addr: %p kanari hossz: %d):\n", m+can_len, (int)can_len);
00135         else
00136             fprintf(fp, "Dump: (addr: %p) \n", m);
00137         size += 2*can_len;
00138         for (s = 0; s < (size+15)/16; s++) {
00139             fprintf(fp, "%04x:%c ", s*16, s*16 < can_len || s*16 >= size-can_len ? ' ' : '*');
00140             for (o = 0; o < 16; o++) {
00141                 if (o == 8) fprintf(fp, " ");
00142                 if (s*16+o < size)
00143                     fprintf(fp, "%02x ", m[s*16+o]);
00144                 else
00145                     fprintf(fp, "   ");
00146             }
00147             fprintf(fp, " ");
00148             for (o = 0; o < 16; o++) {
00149                 if (s*16+o < size)
00150                     fprintf(fp, "%c", isprint(m[s*16+o]) ? m[s*16+o] : '.');
00151                 else
00152                     fprintf(fp, " ");
00153             }
00154             fprintf(fp, "\n");
00155         }
00156     }
00157
00158     void mem_dump(void const *mem, size_t size, FILE* fp) {
00159         dump_memory(mem, size, 0, fp);
00160     }
00161
00162     static BOOL dying;
00163
00164     static void die(const char * msg, void * p, size_t size, call_t * a, call_t * d) {
00165         #ifdef MEMTRACE_ERRFILE
00166             fperror = fopen(XSTR(MEMTRACE_ERRFILE), "w");
00167         #endif
00168         fprintf(fperror,"%s\n",msg);
00169         if (p) {
00170             fprintf(fperror, "\tPointer:\t%p", PU(p));
00171             if (size) fprintf(fperror," (%d byte)", (int)size);
00172             fprintf(fperror,"\n");
00173         }
00174         if (a) print_call("\tFoglalas:\t", *a);
00175         if (d) print_call("\tFelszabaditas:\t", *d);
00176                 if (p) dump_memory(p, size, CANARY_LEN, fperror);
00177
00178         dying = TRUE;
00179         exit(120);
00180     }
00181
00182     static void initialize();
00183 END_NAMESPACE
00184
```

```
00185 /*********************************************************************/
00186 /* MEMTRACE_TO_MEMORY */
00187 /*********************************************************************/
00188
00189 #ifdef MEMTRACE_TO_MEMORY
00190 START_NAMESPACE
00191     typedef struct _registry_item {
00192         void * p;    /* mem pointer*/
00193         size_t size; /* size*/
00194         call_t call;
00195         struct _registry_item * next;
00196     } registry_item;
00197
00198     static registry_item registry; /*sentinel*/
00199
00200     static void print_registry_item(registry_item * p) {
00201         if (p) {
00202             print_registry_item(p->next);
00203             fprintf(fperror, "\t%p%5d byte ",p->p, (int)p->size);
00204             print_call(NULL, p->call);
00205             if(p->call.par_txt) free(p->call.par_txt);
00206             if(p->call.file) free(p->call.file);
00207             free(p);
00208         }
00209     }
00210
00211     /* ha nincs hiba, akkor 0-val tér vissza */
00212     int mem_check(void) {
00213         initialize();
00214         if(dying) return  2;    /* címzési hiba */
00215
00216         if(registry.next) {
00217             /*szivarog*/
00218             #ifdef MEMTRACE_ERRFILE
00219                 fperror = fopen(XSTR(MEMTRACE_ERRFILE), "w");
00220             #endif
00221             fprintf(fperror, "Szivargas:\n");
00222             print_registry_item(registry.next);
00223             registry.next = NULL;
00224             return 1;           /* memória fogyás */
00225         }
00226         return 0;
00227     }
00228 END_NAMESPACE
00229 #endif/*MEMTRACE_TO_MEMORY*/
00230
00231 /*********************************************************************/
00232 /* MEMTRACE_TO_FILE */
00233 /*********************************************************************/
00234
00235 #ifdef MEMTRACE_TO_FILE
00236 START_NAMESPACE
00237     static FILE * trace_file;
00238 END_NAMESPACE
00239 #endif
00240
00241 /*********************************************************************/
00242 /* register/unregister */
00243 /*********************************************************************/
00244
00245 START_NAMESPACE
00246     static int allocated_blks;
00247
00248     int allocated_blocks() { return allocated_blks; }
00249
00250     static BOOL register_memory(void * p, size_t size, call_t call) {
00251         initialize();
00252         allocated_blks++;
00253         #ifdef MEMTRACE_TO_FILE
00254             fprintf(trace_file, "%p\t%d\t%s%s", PU(p), (int)size, pretty[call.f], call.par_txt ?
      call.par_txt : "?");
00255             if (call.f <= 3) fprintf(trace_file, ")");
00256             fprintf(trace_file, "\t%d\t%s\n", call.line, call.file ? call.file : "?");
00257             fflush(trace_file);
00258         #endif
00259         #ifdef MEMTRACE_TO_MEMORY
00260         {/*C-blokk*/
00261             registry_item * n = (registry_item*)malloc(sizeof(registry_item));
00262             if(n==NULL) return FALSE;
00263             n->p = p;
00264             n->size = size;
00265             n->call = call;
00266             n->next = registry.next;
00267             registry.next = n;
00268         }/*C-blokk*/
00269         #endif
00270
```

```
00271         return TRUE;
00272     }
00273
00274     #ifdef MEMTRACE_TO_MEMORY
00275     static registry_item *find_registry_item(void * p) {
00276                 registry_item *n = &registry;
00277                 for(; n->next && n->next->p != p ; n=n->next);
00278                 return n;
00279         }
00280         #endif
00281
00282     static void unregister_memory(void * p, call_t call) {
00283         initialize();
00284         #ifdef MEMTRACE_TO_FILE
00285                         fprintf(trace_file, "%p\t%d\t%s%s", PU(p), -1, pretty[call.f], call.par_txt ?
      call.par_txt : "?");
00286                         if (call.f <= 3) fprintf(trace_file, ")");
00287             fprintf(trace_file,"\t%d\t%s\n",call.line, call.file ? call.file : "?");
00288             fflush(trace_file);
00289         #endif
00290         #ifdef MEMTRACE_TO_MEMORY
00291         { /*C-blokk*/
00292             registry_item * n = find_registry_item(p);
00293             if(n->next) {
00294                 allocated_blks--;
00295                 registry_item * r = n->next;
00296                 n->next = r->next;
00297                 if(COMP(r->call.f,call.f)) {
00298                     int chk = chk_canary(r->p, r->size);
00299                     if (chk < 0)
00300                         die("Blokk elott serult a memoria:", r->p,r->size,&r->call,&call);
00301                     if (chk > 0)
00302                         die("Blokk utan serult a memoria", r->p,r->size,&r->call,&call);
00303                     /*rendben van minden*/
00304                     if(call.par_txt) free(call.par_txt);
00305                     if(r->call.par_txt) free(r->call.par_txt);
00306                     if(call.file) free(call.file);
00307                     if(r->call.file) free(r->call.file);
00308                     memset(PU(r->p), 'f', r->size);
00309                     PU(r->p)[r->size-1] = 0;
00310                     free(r);
00311                 } else {
00312                     /*hibas felszabaditas*/
00313                     die("Hibas felszabaditas:",r->p,r->size,&r->call,&call);
00314                 }
00315             } else {
00316                 die("Nem letezo, vagy mar felszabaditott adat felszabaditasa:", p, 0,NULL,&call);
00317             }
00318         } /*C-blokk*/
00319         #endif
00320     }
00321 END_NAMESPACE
00322
00323 /*********************************************************************/
00324 /* C-stílusú memóriakezelés */
00325 /*********************************************************************/
00326
00327 #ifdef MEMTRACE_C
00328 START_NAMESPACE
00329     void * traced_malloc(size_t size, const char * par_txt, int line, const char * file) {
00330         void * p;
00331         initialize();
00332         p = canary_malloc(size, random_byte);
00333         if (p) {
00334             if(!register_memory(p,size,pack(FMALLOC,par_txt,line,file))) {
00335                 free(p);
00336                 return NULL;
00337             }
00338             return PU(p);
00339         }
00340         return NULL;
00341     }
00342
00343     void * traced_calloc(size_t count, size_t size, const char * par_txt, int line, const char * file)
      {
00344         void * p;
00345         initialize();
00346                 size *= count;
00347                 p = canary_malloc(size, 0);
00348         if(p) {
00349             if(!register_memory(p,size,pack(FCALLOC,par_txt,line,file))) {
00350                 free(p);
00351                 return NULL;
00352             }
00353             return PU(p);
00354         }
00355         return NULL;
```

```
00356     }
00357
00358     void traced_free(void * pu, const char * par_txt, int line, const char * file) {
00359         initialize();
00360         if(pu) {
00361             unregister_memory(P(pu), pack(FFREE,par_txt,line,file));
00362             free(P(pu));
00363         } else {
00364             /*free(NULL) eset*/
00365             #ifdef MEMTRACE_TO_FILE
00366                 fprintf(trace_file,"%s\t%d\t%10s\t","NULL",-1,pretty[FFREE]);
00367                 fprintf(trace_file,"%d\t%s\n",line,file ? file : "?");
00368                 fflush(trace_file);
00369             #endif
00370             #ifndef ALLOW_FREE_NULL
00371             {/*C-blokk*/
00372                 call_t call;
00373                 call = pack(FFREE,par_txt,line,file);
00374                 die("free(NULL) hivasa:",NULL,0,NULL,&call);
00375             }/*C-blokk*/
00376             #endif
00377         }
00378     }
00379
00380     void * traced_realloc(void * old, size_t size, const char * par_txt, int line, const char * file)
    {
00381         void * p;
00382         size_t oldsize = 0;
00383         registry_item * n;
00384         initialize();
00385
00386         #ifdef MEMTRACE_TO_MEMORY
00387                 n = find_registry_item(P(old));
00388                 if (n) oldsize = n->next->size;
00389             p = canary_malloc(size, random_byte);
00390             #else
00391                 p = realloc(old, size);
00392             #endif
00393         if (p) {
00394             /*Ha sikerult a foglalas, regisztraljuk*/
00395             register_memory(p,size,pack(FREALLOC, par_txt, line,file));
00396                     if (old) {
00397                 #ifdef MEMTRACE_TO_MEMORY
00398                     int cpsize = 2*CANARY_LEN;
00399                             if (oldsize < size) cpsize += oldsize;
00400                             else cpsize += size;
00401                             memcpy(p, P(old), cpsize);
00402                 #endif
00403                         unregister_memory(P(old), pack(FREALLOC, par_txt, line, file));
00404                 #ifdef MEMTRACE_TO_MEMORY
00405                             free P(old);
00406                         #endif
00407                     }
00408                     return PU(p);
00409         } else {
00410                 return NULL;
00411             }
00412     }
00413
00414 END_NAMESPACE
00415 #endif/*MEMTRACE_C*/
00416
00417 /*****************************************************************/
00418 /* C++-stílusú memóriakezelés */
00419 /*****************************************************************/
00420
00421 #ifdef MEMTRACE_CPP
00422 START_NAMESPACE
00423     std::new_handler _new_handler;
00424
00425     void _set_new_handler(std::new_handler h) {
00426         initialize();
00427         _new_handler = h;
00428     }
00429
00430     static call_t delete_call;
00431     static BOOL delete_called;
00432
00433     void set_delete_call(int line, const char * file) {
00434         initialize();
00435         delete_call=pack(0,"",line,file); /*func értéke lényegtelen, majd felülírjuk*/
00436         delete_called = TRUE;
00437     }
00438
00439     void * traced_new(size_t size, int line, const char * file, int func) {
00440         initialize();
00441         for (;;) {
```

```
00442             void * p = canary_malloc(size, random_byte);
00443             if(p) {
00444                 register_memory(p,size,pack(func,"",line,file));
00445                 return PU(p);
00446             }
00447 
00448             if (_new_handler == 0)
00449                 throw std::bad_alloc();
00450 
00451             _new_handler();
00452         }
00453     }
00454 
00455     void traced_delete(void * pu, int func) {
00456         initialize();
00457         if(pu) {
00458             /*kiolvasom call-t, ha van*/
00459             memtrace::call_t call = delete_called ? (delete_call.f=func, delete_call) :
      pack(func,NULL,0,NULL);
00460             memtrace::unregister_memory(P(pu),call);
00461             free(P(pu));
00462         }
00463         delete_called=FALSE;
00464     }
00465 END_NAMESPACE
00466 
00467 void * operator new(size_t size, int line, const char * file) THROW_BADALLOC {
00468     return memtrace::traced_new(size,line,file,FNEW);
00469 }
00470 
00471 void * operator new[](size_t size, int line, const char * file) THROW_BADALLOC {
00472     return memtrace::traced_new(size,line,file,FNEWARR);
00473 }
00474 
00475 void * operator new(size_t size) THROW_BADALLOC {
00476     return memtrace::traced_new(size,0,NULL,FNEW);
00477 }
00478 
00479 void * operator new[](size_t size) THROW_BADALLOC {
00480     return memtrace::traced_new(size,0,NULL,FNEWARR);
00481 }
00482 
00483 void operator delete(void * p) THROW_NOTHING {
00484     memtrace::traced_delete(p,FDELETE);
00485 }
00486 
00487 void operator delete[](void * p) THROW_NOTHING {
00488     memtrace::traced_delete(p,FDELETEARR);
00489 }
00490 
00491 
00492 /* Visual C++ 2012 miatt kell, mert háklis, hogy nincs megfelelő delete, bár senki sem használja */
00493 void operator delete(void * p, int, const char *) THROW_NOTHING {
00494     memtrace::traced_delete(p,FDELETE);
00495 }
00496 
00497 void operator delete[](void * p, int, const char *) THROW_NOTHING {
00498     memtrace::traced_delete(p,FDELETE);
00499 }
00500 
00501 #endif/*MEMTRACE_CPP*/
00502 
00503 /*********************************************************************/
00504 /* initialize */
00505 /*********************************************************************/
00506 
00507 START_NAMESPACE
00508     static void initialize() {
00509         static BOOL first = TRUE;
00510         if(first) {
00511             fperror = stderr;
00512             random_byte = (unsigned char)time(NULL);
00513             first = FALSE;
00514             dying = FALSE;
00515             #ifdef MEMTRACE_TO_MEMORY
00516                 registry.next = NULL;
00517                 #if !defined(USE_ATEXIT_OBJECT) && defined(MEMTRACE_AUTO)
00518                     atexit((void(*)(void))mem_check);
00519                 #endif
00520             #endif
00521             #ifdef MEMTRACE_TO_FILE
00522                 trace_file = fopen("memtrace.dump","w");
00523             #endif
00524             #ifdef MEMTRACE_CPP
00525                 _new_handler = NULL;
00526                 delete_called = FALSE;
00527                 delete_call = pack(0,NULL,0,NULL);
```

```
00528             #endif
00529         }
00530     }
00531
00532 #if defined(MEMTRACE_TO_MEMORY) && defined(USE_ATEXIT_OBJECT)
00533     int atexit_class::counter = 0;
00534     int atexit_class::err = 0;
00535 #endif
00536 END_NAMESPACE
00537 #endif
```

5.11 memtrace.h

```
00001 /*********************************
00002 Memoriaszivargas-detektor
00003 Keszitette: Peregi Tamas, BME IIT, 2011
00004             petamas@iit.bme.hu
00005 Kanari:     Szeberenyi Imre, 2013.,
00006 VS 2012:    Szeberényi Imre, 2015.,
00007 mem_dump:   2016.
00008 inclue-ok:  2017., 2018. 2019.
00009 *********************************/
00010
00011 #ifndef MEMTRACE_H
00012 #define MEMTRACE_H
00013
00014 #if defined(MEMTRACE)
00015
00016 /*ha definiálva van, akkor a hibakat ebbe a fajlba írja, egyébkent stderr-re*/
00017 /*#define MEMTRACE_ERRFILE MEMTRACE.ERR*/
00018
00019 /*ha definialva van, akkor futas kozben lancolt listat epit. Javasolt a hasznalata*/
00020 #define MEMTRACE_TO_MEMORY
00021
00022 /*ha definialva van, akkor futas kozben fajlba irja a foglalasokat*/
00023 /*ekkor nincs ellenorzes, csak naplozas*/
00024 /*#define MEMTRACE_TO_FILE*/
00025
00026 /*ha definialva van, akkor a megallaskor automatikus riport keszul */
00027 #define MEMTRACE_AUTO
00028
00029 /*ha definialva van, akkor malloc()/calloc()/realloc()/free() kovetve lesz*/
00030 #define MEMTRACE_C
00031
00032 #ifdef MEMTRACE_C
00033     /*ha definialva van, akkor free(NULL) nem okoz hibat*/
00034     #define ALLOW_FREE_NULL
00035 #endif
00036
00037 #ifdef __cplusplus
00038     /*ha definialva van, akkor new/delete/new[]/delete[] kovetve lesz*/
00039     #define MEMTRACE_CPP
00040 #endif
00041
00042 #if defined(__cplusplus) && defined(MEMTRACE_TO_MEMORY)
00043     /*ha definialva van, akkor atexit helyett objektumot hasznal*/
00044     /*ajanlott bekapcsolni*/
00045     #define USE_ATEXIT_OBJECT
00046 #endif
00047
00048 /*****************************************/
00049 /* INNEN NE MODOSITSD                    */
00050 /*****************************************/
00051 #ifdef NO_MEMTRACE_TO_FILE
00052     #undef MEMTRACE_TO_FILE
00053 #endif
00054
00055 #ifdef NO_MEMTRACE_TO_MEMORY
00056     #undef MEMTRACE_TO_MEMORY
00057 #endif
00058
00059 #ifndef MEMTRACE_AUTO
00060     #undef USE_ATEXIT_OBJECT
00061 #endif
00062
00063 #ifdef __cplusplus
00064     #define START_NAMESPACE namespace memtrace {
00065     #define END_NAMESPACE } /*namespace*/
00066     #define TRACEC(func) memtrace::func
00067     #include <new>
00068 #else
00069     #define START_NAMESPACE
00070     #define END_NAMESPACE
```

```
00071     #define TRACEC(func) func
00072 #endif
00073
00074 // THROW deklaráció változatai
00075 #if defined(_MSC_VER)
00076   // VS rosszul kezeli az __cplusplus makrot
00077   #if _MSC_VER < 1900
00078     // * nem biztos, hogy jó így *
00079     #define THROW_BADALLOC
00080     #define THROW_NOTHING
00081   #else
00082     // C++11 vagy újabb
00083     #define THROW_BADALLOC noexcept(false)
00084     #define THROW_NOTHING noexcept
00085   #endif
00086 #else
00087   #if __cplusplus < 201103L
00088     // C++2003 vagy régebbi
00089     #define THROW_BADALLOC throw (std::bad_alloc)
00090     #define THROW_NOTHING throw ()
00091   #else
00092     // C++11 vagy újabb
00093     #define THROW_BADALLOC noexcept(false)
00094     #define THROW_NOTHING noexcept
00095   #endif
00096 #endif
00097
00098 START_NAMESPACE
00099     int allocated_blocks();
00100 END_NAMESPACE
00101
00102 #if defined(MEMTRACE_TO_MEMORY)
00103 START_NAMESPACE
00104         int mem_check(void);
00105 END_NAMESPACE
00106 #endif
00107
00108 #if defined(MEMTRACE_TO_MEMORY) && defined(USE_ATEXIT_OBJECT)
00109 #include <cstdio>
00110 START_NAMESPACE
00111     class atexit_class {
00112         private:
00113             static int counter;
00114             static int err;
00115         public:
00116             atexit_class() {
00117 #if defined(CPORTA) && !defined(CPORTA_NOSETBUF)
00118                 if (counter == 0) {
00119                         setbuf(stdout, 0);
00120                         setbuf(stderr, 0);
00121                 }
00122 #endif
00123             counter++;
00124             }
00125
00126             int check() {
00127                 if(--counter == 0)
00128                     err = mem_check();
00129                 return err;
00130             }
00131
00132             ~atexit_class() {
00133                 check();
00134             }
00135     };
00136
00137 static atexit_class atexit_obj;
00138
00139 END_NAMESPACE
00140 #endif/*MEMTRACE_TO_MEMORY && USE_ATEXIT_OBJECT*/
00141
00142 /*Innentol csak a "normal" include eseten kell, kulonben osszezavarja a mukodest*/
00143 #ifndef FROM_MEMTRACE_CPP
00144 #include <stdlib.h>
00145 #ifdef __cplusplus
00146     #include <iostream>
00147 /* ide gyűjtjük a nemtrace-vel összeakadó headereket, hogy előbb legyenek */
00148
00149     #include <fstream>  // VS 2013 headerjében van deleted definició
00150     #include <sstream>
00151     #include <vector>
00152     #include <list>
00153     #include <map>
00154     #include <algorithm>
00155     #include <functional>
00156 #endif
00157 #ifdef MEMTRACE_CPP
```

```
00158     namespace std {
00159         typedef void (*new_handler)();
00160 }
00161 #endif
00162
00163 #ifdef MEMTRACE_C
00164 START_NAMESPACE
00165     #undef malloc
00166     #define malloc(size) TRACEC(traced_malloc)(size,#size,__LINE__,__FILE__)
00167     void * traced_malloc(size_t size, const char *size_txt, int line, const char * file);
00168
00169     #undef calloc
00170     #define calloc(count,size) TRACEC(traced_calloc)(count, size, #count","#size,__LINE__,__FILE__)
00171     void * traced_calloc(size_t count, size_t size, const char *size_txt, int line, const char *
      file);
00172
00173     #undef free
00174     #define free(p) TRACEC(traced_free)(p, #p,__LINE__,__FILE__)
00175     void traced_free(void * p, const char *size_txt, int line, const char * file);
00176
00177     #undef realloc
00178     #define realloc(old,size) TRACEC(traced_realloc)(old,size,#size,__LINE__,__FILE__)
00179     void * traced_realloc(void * old, size_t size, const char *size_txt, int line, const char * file);
00180
00181     void mem_dump(void const *mem, size_t size, FILE* fp);
00182
00183
00184 END_NAMESPACE
00185 #endif/*MEMTRACE_C*/
00186
00187 #ifdef MEMTRACE_CPP
00188 START_NAMESPACE
00189     #undef set_new_handler
00190     #define set_new_handler(f) TRACEC(_set_new_handler)(f)
00191     void _set_new_handler(std::new_handler h);
00192
00193     void set_delete_call(int line, const char * file);
00194 END_NAMESPACE
00195
00196 void * operator new(size_t size, int line, const char * file) THROW_BADALLOC;
00197 void * operator new[](size_t size, int line, const char * file) THROW_BADALLOC;
00198 void * operator new(size_t size) THROW_BADALLOC;
00199 void * operator new[](size_t size) THROW_BADALLOC;
00200 void operator delete(void * p)  THROW_NOTHING;
00201 void operator delete[](void * p) THROW_NOTHING;
00202
00203 /* Visual C++ 2012 miatt kell, mert háklis, hogy nincs megfelelő delete, bár senki sem használja */
00204 void operator delete(void *p, int, const char *) THROW_NOTHING;
00205 void operator delete[](void *p, int, const char *) THROW_NOTHING;
00206
00207
00208 #define new new(__LINE__, __FILE__)
00209 #define delete memtrace::set_delete_call(__LINE__, __FILE__),delete
00210
00211 #ifdef CPORTA
00212 #define system(...)  // system(__VA_ARGS__)
00213 #endif
00214
00215 #endif /*MEMTRACE_CPP*/
00216
00217 #endif /*FROM_MEMTRACE_CPP*/
00218 #endif /*MEMCHECK*/
00219 #endif /*MEMTRACE_H*/
```

5.12 menu.cpp

```
00001 #include <iostream>
00002 #include <cstring>
00003 #include <cmath>
00004 #include "csapat.h"
00005 #include "kezi.h"
00006 #include "kosar.h"
00007 #include "foci.h"
00008 #include "nyilvantartas.h"
00009 #include "menu.h"
00010 #include "memtrace.h"
00011
00012 using std::cout;
00013 using std::cin;
00014 using std::endl;
00015
00016 Menu::Menu(){
00017   DB.load();
```

```
00018   cout « "[+] Fileok Betoltese..." « endl;
00019 }
00020
00021 Menu::~Menu(){
00022   DB.save();
00023   cout « "\n[+] Mentes...\n" « endl;
00024 }
00025
00026 int Menu::maxStdRowLen() const {
00027   // A leghoszabb sorszám kiszámolása (teljessen folosleges es lassitja a programot, de szep lesz):
00028   int maxrow = 0;
00029   for (KeziListaElem* i = DB.getKeziLista(); i != nullptr; i = i->kovi ) {
00030     int currRow = strlen(i->adat.getNev()) + Nyilvantartas::intlen(i->adat.getLetszam())
00031       + Nyilvantartas::intlen(i->adat.getTamogatas()) + 4 + 4;
00032     if (currRow > maxrow) { maxrow = currRow;}
00033   }
00034
00035   for (KosarListaElem* i = DB.getKosarLista(); i != nullptr; i = i->kovi ) {
00036     int currRow = strlen(i->adat.getNev()) + Nyilvantartas::intlen(i->adat.getLetszam())
00037       + Nyilvantartas::intlen(i->adat.getPomPomDb()) + 4 + 4;
00038     if (currRow > maxrow) { maxrow = currRow;}
00039   }
00040
00041   for (FociListaElem* i = DB.getFociLista(); i != nullptr; i = i->kovi ) {
00042     int currRow = strlen(i->adat.getNev()) + Nyilvantartas::intlen(i->adat.getLetszam())
00043       + Nyilvantartas::intlen(i->adat.getEdzokszama()) + 4 + 4;
00044     if (currRow > maxrow) { maxrow = currRow;}
00045   }
00046   return maxrow;
00047 }
00048
00049 int Menu::maxStdRowLen(Tipus t) const {
00050   // A leghoszabb sorszám kiszámolása (teljessen folosleges es lassitja a programot, de szep lesz):
00051   int maxrow = 0;
00052   if (t == KEZI) {
00053     for (KeziListaElem* i = DB.getKeziLista(); i != nullptr; i = i->kovi ) {
00054       int currRow = strlen(i->adat.getNev()) + Nyilvantartas::intlen(i->adat.getLetszam())
00055         + Nyilvantartas::intlen(i->adat.getTamogatas()) + 4 + 4;
00056       if (currRow > maxrow) { maxrow = currRow;}
00057     }
00058   }
00059   if (t == KOSAR){
00060     for (KosarListaElem* i = DB.getKosarLista(); i != nullptr; i = i->kovi ) {
00061       int currRow = strlen(i->adat.getNev()) + Nyilvantartas::intlen(i->adat.getLetszam())
00062         + Nyilvantartas::intlen(i->adat.getPomPomDb()) + 4 + 4;
00063       if (currRow > maxrow) { maxrow = currRow;}
00064     }
00065   }
00066   if (t == FOCI) {
00067     for (FociListaElem* i = DB.getFociLista(); i != nullptr; i = i->kovi ) {
00068       int currRow = strlen(i->adat.getNev()) + Nyilvantartas::intlen(i->adat.getLetszam())
00069         + Nyilvantartas::intlen(i->adat.getEdzokszama()) + 4 + 4;
00070       if (currRow > maxrow) { maxrow = currRow;}
00071     }
00072   }
00073   return maxrow;
00074 }
00075
00076 int Menu::getStdRowLen(KeziListaElem *l) const {
00077   int n = 0;
00078   n += strlen(l->adat.getNev());
00079   n += Nyilvantartas::intlen(l->adat.getLetszam());
00080   n += Nyilvantartas::intlen(l->adat.getTamogatas());
00081   return n;
00082 }
00083
00084 int Menu::getStdRowLen(KosarListaElem *l) const {
00085   int n = 0;
00086   n += strlen(l->adat.getNev());
00087   n += Nyilvantartas::intlen(l->adat.getLetszam());
00088   n += Nyilvantartas::intlen(l->adat.getPomPomDb());
00089   return n;
00090 }
00091
00092 int Menu::getStdRowLen(FociListaElem *l) const {
00093   int n = 0;
00094   n += strlen(l->adat.getNev());
00095   n += Nyilvantartas::intlen(l->adat.getLetszam());
00096   n += Nyilvantartas::intlen(l->adat.getEdzokszama());
00097   return n;
00098 }
00099
00100 void Menu::printAll() const {
00101   int maxrow = maxStdRowLen();
00102   if (DB.getKeziLista() == nullptr) {
00103     cout « "\n\n[+] Ures a kezilabda csapatok listaja\n" « endl;
00104   } else {
```

```
00105     cout « "\n\n\n[+] Kezilabda Csapatok:" « endl « endl;
00106     cout « "Nev-Letszam-Tamogatas"« endl;
00107     for (int i=0;i<maxrow;i++) {cout « "=";} cout « endl;
00108     for (KeziListaElem* i = DB.getKeziLista(); i != nullptr; i = i->kovi) {
00109       int t = abs(maxrow-6 - getStdRowLen(i));
00110       //cout « "\t\t---" « t « "---" « maxrow « "---" « getStdRowLen(i) « endl;
00111       cout « i->adat.getNev();
00112       for (int j=0; j<t; j++) {cout « " " ;}
00113       cout « i->adat.getLetszam() « "\t" « i->adat.getTamogatas() « endl;
00114     }
00115     for (int i=0;i<maxrow;i++) {cout « "=";} cout « endl;
00116   }
00117
00118   if (DB.getKosarLista() == nullptr) {
00119     cout « "\n\n[+] Ures a kosarlabda csapatok listaja\n" « endl;
00120   } else {
00121     cout « "\n\n\n[+] Kosarlabda Csapatok:" « endl « endl;
00122     cout « "Nev-Letszam-PomPomLanyok"« endl;
00123     for (int i=0;i<maxrow;i++) {cout « "=";} cout « endl;
00124     for (KosarListaElem* i = DB.getKosarLista(); i != nullptr; i = i->kovi) {
00125       int t = abs(maxrow-6 - getStdRowLen(i));;
00126       //cout « "\t\t---" « t « "---" « maxrow « "---" « getStdRowLen(i) « endl;
00127       cout « i->adat.getNev();
00128       for (int j=0; j<t; j++) {cout « " " ;}
00129       cout « i->adat.getLetszam() « "\t" « i->adat.getPomPomDb() « endl;
00130     }
00131     for (int i=0;i<maxrow;i++) {cout « "=";} cout « endl;
00132   }
00133
00134   if (DB.getFociLista() == nullptr) {
00135     cout « "\n\n[+] Ures a focilabda csapatok listaja\n" « endl;
00136   } else {
00137     cout « "\n\n\n[+] Focilabda Csapatok:" « endl « endl;
00138     cout « "Nev-Letszam-Edzok"« endl;
00139     for (int i=0;i<maxrow;i++) {cout « "=";} cout « endl;
00140     for (FociListaElem* i = DB.getFociLista(); i != nullptr; i = i->kovi) {
00141       int t = abs(maxrow-6 - getStdRowLen(i));;
00142       //cout « "\t\t---" « t « "---" « maxrow « "---" « getStdRowLen(i) « endl;
00143       cout « i->adat.getNev();
00144       for (int j=0; j<t; j++) {cout « " " ;}
00145       cout « i->adat.getLetszam() « "\t" « i->adat.getEdzokszama() « endl;
00146     }
00147
00148     for (int i=0;i<maxrow;i++) {cout « "=";} cout « endl;
00149   }
00150 }
00151
00152 void Menu::printKezi() const {
00153   if (DB.getKeziLista() == nullptr) { cout « "\n\n[+]Ures a kezilabda csapatok listaja\n\n" « endl;
      return; }
00154   int maxrow = maxStdRowLen(KEZI);
00155   cout « "\n\n\n\t[+] Kezilabda Csapatok:" « endl « endl;
00156   cout « "\tNev-Letszam-Tamogatas"« endl;
00157   cout«"\t"; for (int i=0;i<maxrow;i++) {cout « "=";} cout « endl;
00158   for (KeziListaElem* i = DB.getKeziLista(); i != nullptr; i = i->kovi) {
00159     int t = abs(maxrow-6 - getStdRowLen(i));
00160     //cout « "\t\t---" « t « "---" « maxrow « "---" « getStdRowLen(i) « endl;
00161     cout « "\t" « i->adat.getNev();
00162     for (int j=0; j<t; j++) {cout « " " ;}
00163     cout « i->adat.getLetszam() « "\t" « i->adat.getTamogatas() « endl;
00164   }
00165   cout«"\t"; for (int i=0;i<maxrow;i++) {cout « "=";} cout « endl;
00166 }
00167
00168 void Menu::printKosar() const {
00169   if (DB.getKosarLista() == nullptr) { cout « "\n\n[+]Ures a kosarlabda csapatok listaja\n\n" « endl;
      return; }
00170   int maxrow = maxStdRowLen(KOSAR);
00171   cout « "\n\n\n\t[+] Kosarlabda Csapatok:" « endl « endl;
00172   cout « "\tNev-Letszam-PomPomLanyok"« endl;
00173   cout«"\t"; for (int i=0;i<maxrow;i++) {cout « "=";} cout « endl;
00174   for (KosarListaElem* i = DB.getKosarLista(); i != nullptr; i = i->kovi) {
00175     int t = abs(maxrow-6 - getStdRowLen(i));
00176     //cout « "\t\t---" « t « "---" « maxrow « "---" « getStdRowLen(i) « endl;
00177     cout « "\t" « i->adat.getNev();
00178     for (int j=0; j<t; j++) {cout « " " ;}
00179     cout « i->adat.getLetszam() « "\t" « i->adat.getPomPomDb() « endl;
00180   }
00181   cout«"\t"; for (int i=0;i<maxrow;i++) {cout « "=";} cout « endl;
00182 }
00183
00184 void Menu::printFoci() const {
00185   if (DB.getFociLista() == nullptr) { cout « "\n\n[+]Ures a focilabda csapatok listaja\n\n" « endl;
      return; }
00186   int maxrow = maxStdRowLen(FOCI);
00187   cout « "\n\n\n\t[+] Focilabda Csapatok:" « endl « endl;
00188   cout « "\tNev-Letszam-EdzokSzama"« endl;
```

```
00189   cout«"\t"; for (int i=0;i<maxrow;i++) {cout « "=";} cout « endl;
00190   for (FociListaElem* i = DB.getFociLista(); i != nullptr; i = i->kovi) {
00191     int t = abs(maxrow-6 - getStdRowLen(i));
00192     //cout « "\t\t---" « t « "---" « maxrow « "---" « getStdRowLen(i) « endl;
00193     cout « "\t" « i->adat.getNev();
00194     for (int j=0; j<t; j++) {cout « " " ;}
00195     cout « i->adat.getLetszam() « "\t" « i->adat.getEdzokszama() « endl;
00196   }
00197   cout«"\t"; for (int i=0;i<maxrow;i++) {cout « "=";} cout « endl;
00198 }
00199
00200 void Menu::printOne(KeziListaElem *p) const {
00201   if (p == nullptr) {return;}
00202   int sorlen = getStdRowLen(p);
00203   cout « endl « endl;
00204   cout « "\t\t[+] KERESETT KEZI CSAPAT" « endl;
00205   cout « "\t\tNev-Letszam-Tamogatas" « endl;
00206   cout « "\t\t"; for (int i=0; i<sorlen+10;i++){cout « "=";} cout « endl;
00207   cout « "\t\t" « p->adat.getNev() « "\t" « p->adat.getLetszam() « "\t"
00208        « p->adat.getTamogatas() « endl;
00209   cout « "\t\t"; for (int i=0; i<sorlen+10;i++){cout « "=";} cout « endl « endl;
00210 }
00211
00212 void Menu::printOne(KosarListaElem *p) const {
00213   if (p == nullptr) {return;}
00214   int sorlen = getStdRowLen(p);
00215   cout « endl « endl;
00216   cout « "\t\t[+] KERESETT KOSAR CSAPAT" « endl;
00217   cout « "\t\tNev-Letszam-PomPomLanyok" « endl;
00218   cout « "\t\t"; for (int i=0; i<sorlen+10;i++){cout « "=";} cout « endl;
00219   cout « "\t\t" « p->adat.getNev() « "\t" « p->adat.getLetszam() « "\t"
00220        « p->adat.getPomPomDb() « endl;
00221   cout « "\t\t"; for (int i=0; i<sorlen+10;i++){cout « "=";} cout « endl « endl;
00222 }
00223
00224 void Menu::printOne(FociListaElem *p) const {
00225   if (p == nullptr) {return;}
00226   int sorlen = getStdRowLen(p);
00227   cout « endl « endl;
00228   cout « "\t\t[+] KERESETT FOCICSAPAT" « endl;
00229   cout « "\t\tNev-Letszam-Edzok" « endl;
00230   cout « "\t\t"; for (int i=0; i<sorlen+10;i++){cout « "=";} cout « endl;
00231   cout « "\t\t" « p->adat.getNev() « "\t" « p->adat.getLetszam() « "\t"
00232        « p->adat.getEdzokszama() « endl;
00233   cout « "\t\t"; for (int i=0; i<sorlen+10;i++){cout « "=";} cout « endl « endl;
00234 }
00235
00236 void Menu::keziMenu() {
00237   int arg = -1;
00238   while (arg != 0) {
00239     cout « "\n\n\t[+]KEZILABDA CSAPAT MENU" « endl;
00240     cout «
      "\t=============================================================================================="
      « endl;
00241     cout « "\t| (1): Kezis csapatok listazasa | (2): Csapat Keresese | (3): Uj csapat felvetele | (0):
      Vissza |" « endl;
00242     cout «
      "\t=============================================================================================="
      « endl;
00243     cout « "\t[!] Valasszon a listabol. Adja meg a menu szamat!" « endl;
00244     cout « "\t[?] Menu szama: ";
00245     cin.sync();
00246     cin » arg;
00247     switch (arg)
00248     {
00249     case 1:
00250       printKezi();
00251       break;
00252
00253     case 2:
00254       keresKeziMenu();
00255       break;
00256
00257     case 3:
00258       ujMenu(KEZI);
00259       break;
00260
00261     case 0:
00262       arg = 0;
00263       break;
00264
00265     default:
00266       cout « "\n\t[!] Nem letezo menu..." « endl;
00267       break;
00268     }
00269   }
00270 }
```

```
00271
00272 void Menu::kosarMenu() {
00273   int arg = -1;
00274   while (arg != 0) {
00275     cout « "\n\n\t[+]KOSARLABDA CSAPAT MENU" « endl;
00276     cout «
      "\t=============================================================================================================="
      « endl;
00277     cout « "\t| (1): Kosaras csapatok listazasa | (2): Csapat Keresese | (3): Uj csapat felvetele |
      (0): Vissza |" « endl;
00278     cout «
      "\t=============================================================================================================="
      « endl;
00279     cout « "\t[!] Valasszon a listabol. Adja meg a menu szamat!" « endl;
00280     cout « "\t[?] Menu szama: ";
00281     cin.sync();
00282     cin » arg;
00283     switch (arg)
00284     {
00285     case 1:
00286       printKosar();
00287       break;
00288
00289     case 2:
00290       keresKosarMenu();
00291       break;
00292
00293     case 3:
00294       ujMenu(KOSAR);
00295       break;
00296
00297     case 0:
00298       arg = 0;
00299       break;
00300
00301     default:
00302       cout « "\n\t[!] Nem letezo menu..." « endl;
00303       break;
00304     }
00305   }
00306 }
00307
00308 void Menu::fociMenu() {
00309   int arg = -1;
00310   while (arg != 0) {
00311     cout « "\n\n\t[+]FOCILABDA CSAPAT MENU" « endl;
00312     cout «
      "\t=============================================================================================================" «
      endl;
00313     cout « "\t| (1): Focicsapatok listazasa | (2): Csapat Keresese | (3): Uj csapat felvetele | (0):
      Vissza |" « endl;
00314     cout «
      "\t=============================================================================================================" «
      endl;
00315     cout « "\t[!] Valasszon a listabol. Adja meg a menu szamat!" « endl;
00316     cout « "\t[?] Menu szama: ";
00317     cin.sync();
00318     cin » arg;
00319     switch (arg)
00320     {
00321     case 1:
00322       printFoci();
00323       break;
00324
00325     case 2:
00326       keresFociMenu();
00327       break;
00328
00329     case 3:
00330       ujMenu(FOCI);
00331       break;
00332
00333     case 0:
00334       arg = 0;
00335       break;
00336
00337     default:
00338       cout « "\n\t[!] Nem letezo menu..." « endl;
00339       break;
00340     }
00341   }
00342 }
00343
00344 void Menu::keresKeziMenu(){
00345   char buffer[256] = "";
00346   cout « "\t[?] Keresett csapat neve: ";
00347   cin.sync();
```

```
00348   #ifdef __linux__
00349     cin.ignore(1, '\n');
00350   #endif
00351   #ifdef __APPLE__
00352     cin.ignore(1, '\n');
00353   #endif
00354   cin.getline(buffer, 255);
00355   KeziListaElem *result = DB.findKezi(buffer);
00356   if (result == nullptr) {
00357     cout « "[!] Nincs ilyen nevu csapat!" « endl;
00358     return;
00359   } else {
00360     printOne(result);
00361     int arg = -1;
00362     int d = 0;
00363     while (arg != 0) {
00364       cout « "\t\t[+] CSAPAT KEZELO MENU" « endl;
00365       cout « "\t\t=============================================================" « endl;
00366       cout « "\t\t| (1): Csapat modositasa | (2): Csapat torlese | (0): Vissza |" « endl;
00367       cout « "\t\t=============================================================" « endl;
00368       cout « "\t\t[!] Valasszon a listabol. Adja meg a menu szamat!" « endl;
00369       cout « "\t\t[?] Menu szama: ";
00370       cin.sync();
00371       cin » arg;
00372       switch (arg) {
00373         case 1:
00374           editKeziMenu(result);
00375           return;
00376           break;
00377         case 2:
00378           cout « "\n\t\t[?] Biztosan torli a csapatot? [(1): Igen | (0): Nem]: ";
00379           cin.sync();
00380           cin » d;
00381           if (d == 1) {
00382             DB.rm(result);
00383             arg = 0;
00384             cout « "\n[+] Mentes..." « endl;
00385             DB.save();
00386             return;
00387           }
00388           break;
00389         case 0:
00390           arg = 0;
00391           break;
00392         default:
00393           cout « "\n\t[!] Nem letezo menu..." « endl;
00394           break;
00395       }
00396     }
00397   }
00398 }
00399
00400 void Menu::keresKosarMenu(){
00401   char buffer[256] = "";
00402   cout « "\t[?] Keresett csapat neve: ";
00403   cin.sync();
00404   #ifdef __linux__
00405     cin.ignore(1, '\n');
00406   #endif
00407   #ifdef __APPLE__
00408     cin.ignore(1, '\n');
00409   #endif
00410   cin.getline(buffer, 255);
00411   KosarListaElem *result = DB.findKosar(buffer);
00412   if (result == nullptr) {
00413     cout « "[!] Nincs ilyen nevu csapat!" « endl;
00414     return;
00415   } else {
00416     printOne(result);
00417     int arg = -1;
00418     int d = 0;
00419     while (arg != 0) {
00420       cout « "\t\t[+] CSAPAT KEZELO MENU" « endl;
00421       cout « "\t\t=============================================================" « endl;
00422       cout « "\t\t| (1): Csapat modositasa | (2): Csapat torlese | (0): Vissza |" « endl;
00423       cout « "\t\t=============================================================" « endl;
00424       cout « "\t\t[!] Valasszon a listabol. Adja meg a menu szamat!" « endl;
00425       cout « "\t\t[?] Menu szama: ";
00426       cin.sync();
00427       cin » arg;
00428       switch (arg) {
00429         case 1:
00430           editKosarMenu(result);
00431           return;
00432           break;
00433         case 2:
00434           cout « "\n\t\t[?] Biztosan torli a csapatot? [(1): Igen | (0): Nem]: ";
```

```
00435           cin.sync();
00436           cin » d;
00437           if (d == 1) {
00438             DB.rm(result);
00439             arg = 0;
00440             cout « "\n[+] Mentes..." « endl;
00441             DB.save();
00442             return;
00443           }
00444           break;
00445         case 0:
00446           arg = 0;
00447           break;
00448         default:
00449           cout « "\n\t[!] Nem letezo menu..." « endl;
00450           break;
00451       }
00452     }
00453   }
00454 }
00455
00456 void Menu::keresFociMenu(){
00457   char buffer[256] = "";
00458   cout « "\t[?] Keresett csapat neve: ";
00459   cin.sync();
00460   #ifdef __linux__
00461     cin.ignore(1, '\n');
00462   #endif
00463   #ifdef __APPLE__
00464     cin.ignore(1, '\n');
00465   #endif
00466   cin.getline(buffer, 255);
00467   FociListaElem *result = DB.findFoci(buffer);
00468   if (result == nullptr) {
00469     cout « "[!] Nincs ilyen nevu csapat!" « endl;
00470     return;
00471   } else {
00472     printOne(result);
00473     int arg = -1;
00474     int d = 0;
00475     while (arg != 0) {
00476       cout « "\t\t[+] CSAPAT KEZELO MENU" « endl;
00477       cout « "\t\t=================================================================" « endl;
00478       cout « "\t\t| (1): Csapat modositasa | (2): Csapat torlese | (0): Vissza |" « endl;
00479       cout « "\t\t=================================================================" « endl;
00480       cout « "\t\t[!] Valasszon a listabol. Adja meg a menu szamat!" « endl;
00481       cout « "\t\t[?] Menu szama: ";
00482       cin.sync();
00483       cin » arg;
00484       switch (arg) {
00485         case 1:
00486           editFociMenu(result);
00487           return;
00488           break;
00489         case 2:
00490           cout « "\n\t\t[?] Biztosan torli a csapatot? [(1): Igen | (0): Nem]: ";
00491           cin.sync();
00492           cin » d;
00493           if (d == 1) {
00494             DB.rm(result);
00495             arg = 0;
00496             cout « "\n[+] Mentes..." « endl;
00497             DB.save();
00498             return;
00499           }
00500           break;
00501         case 0:
00502           arg = 0;
00503           break;
00504         default:
00505           cout « "\n\t[!] Nem letezo menu..." « endl;
00506           break;
00507       }
00508     }
00509   }
00510 }
00511
00512 void Menu::editKeziMenu(KeziListaElem *p) {
00513   if (p == nullptr) {return;}
00514   cout « endl;
00515   cout « "\t\t\tMODOSITO MENU" « endl;
00516   cout «
      "\t\t\t=============================================================================================" «
      endl;
00517   cout « "\t\t\t| (1): Csapatnev csere | (2): Letszam modositasa | (3): Tamogatas hozzaadasa | (0):
      Vissza |" « endl;
00518   cout «
```

```
      "\t\t\t=============================================================================================" «
      endl;
00519   cout « "\t\t\t[!] Valasszon a listabol. Adja meg a menu szamat!" « endl;
00520   cout « "\t\t\t[?] Menu szama: ";
00521   int arg = -1;
00522   int l = 0;
00523   char buffer[256] = "";
00524   long long int t = 0;
00525   cin.sync();
00526   cin » arg;
00527   while (arg != 0) {
00528     switch (arg) {
00529       case 1:
00530         cout « "\n\t\t\t[?] Uj csapatnev: ";
00531         cin.sync();
00532         #ifdef __linux__
00533           cin.ignore(1, '\n');
00534         #endif
00535         #ifdef __APPLE__
00536           cin.ignore(1, '\n');
00537         #endif
00538         cin.getline(buffer, 255);
00539         p->adat.setNev(buffer);
00540         DB.save();
00541         cout « "\n[+]Mentes..." « endl;
00542         return;
00543         break;
00544
00545       case 2:
00546         cout « "\n\t\t\t[?] Uj Letszam: ";
00547         cin.sync();
00548         cin » l;
00549         p->adat.setLetszam(l);
00550         DB.save();
00551         cout « "\n[+]Mentes..." « endl;
00552         return;
00553         break;
00554
00555       case 3:
00556         cout « "\n\t\t\t[?] Tamogatas hozzaadasa: ";
00557         cin.sync();
00558         cin » t;
00559         p->adat.addTamogatas(t);
00560         DB.save();
00561         cout « "\n[+]Mentes..." « endl;
00562         return;
00563         break;
00564
00565       case 0:
00566         arg = 0;
00567         break;
00568
00569       default:
00570         cout « "\t\t\t[!] Nem letezo menu..." « endl;
00571         break;
00572     }
00573   }
00574 }
00575
00576 void Menu::editKosarMenu(KosarListaElem *p) {
00577   if (p == nullptr) {return;}
00578   cout « endl;
00579   cout « "\t\t\tMODOSITO MENU" « endl;
00580   cout «
      "\t\t\t=============================================================================================="
      « endl;
00581   cout « "\t\t\t| (1): Csapatnev csere | (2): Letszam modositasa | (3): PomPomlanyok hozzaadaasa |
      (0): Vissza |" « endl;
00582   cout «
      "\t\t\t=============================================================================================="
      « endl;
00583   cout « "\t\t\t[!] Valasszon a listabol. Adja meg a menu szamat!" « endl;
00584   cout « "\t\t\t[?] Menu szama: ";
00585   int arg = -1;
00586   int l = 0;
00587   char buffer[256] = "";
00588   int o = 0;
00589   cin.sync();
00590   cin » arg;
00591   while (arg != 0) {
00592     switch (arg) {
00593       case 1:
00594         cout « "\n\t\t\t[?] Uj csapatnev: ";
00595         cin.sync();
00596         #ifdef __linux__
00597           cin.ignore(1, '\n');
00598         #endif
```

```
00599         #ifdef __APPLE__
00600           cin.ignore(1, '\n');
00601         #endif
00602         cin.getline(buffer, 255);
00603         p->adat.setNev(buffer);
00604         DB.save();
00605         cout « "\n[+]Mentes..." « endl;
00606         return;
00607         break;
00608
00609       case 2:
00610         cout « "\n\t\t\t[?] Uj Letszam: ";
00611         cin.sync();
00612         cin » l;
00613         p->adat.setLetszam(l);
00614         DB.save();
00615         cout « "\n[+]Mentes..." « endl;
00616         return;
00617         break;
00618
00619       case 3:
00620         cout « "\n\t\t\t[?] PomPom lanyok hozzaadasa: ";
00621         cin.sync();
00622         cin » o;
00623         p->adat.addPompom(o);
00624         DB.save();
00625         cout « "\n[+]Mentes..." « endl;
00626         return;
00627         break;
00628
00629       case 0:
00630         arg = 0;
00631         break;
00632
00633       default:
00634         cout « "\t\t\t[!] Nem letezo menu..." « endl;
00635         break;
00636     }
00637   }
00638 }
00639
00640 void Menu::editFociMenu(FociListaElem *p) {
00641   if (p == nullptr) {return;}
00642   cout « endl;
00643   cout « "\t\t\tMODOSITO MENU" « endl;
00644   cout «
      "\t\t\t=============================================================================================" «
      endl;
00645   cout « "\t\t\t| (1): Csapatnev csere | (2): Letszam modositasa | (3): Edzok hozzaadasa | (0): Vissza
      |" « endl;
00646   cout «
      "\t\t\t=============================================================================================" «
      endl;
00647   cout « "\t\t\t[!] Valasszon a listabol. Adja meg a menu szamat!" « endl;
00648   cout « "\t\t\t[?] Menu szama: ";
00649   int arg = -1;
00650   int l = 0;
00651   char buffer[256] = "";
00652   int e = 0;
00653   cin.sync();
00654   cin » arg;
00655   while (arg != 0) {
00656     switch (arg) {
00657       case 1:
00658         cout « "\n\t\t\t[?] Uj csapatnev: ";
00659         cin.sync();
00660         #ifdef __linux__
00661           cin.ignore(1, '\n');
00662         #endif
00663         #ifdef __APPLE__
00664           cin.ignore(1, '\n');
00665         #endif
00666         cin.getline(buffer, 255);
00667         p->adat.setNev(buffer);
00668         DB.save();
00669         cout « "\n[+]Mentes..." « endl;
00670         return;
00671         break;
00672
00673       case 2:
00674         cout « "\n\t\t\t[?] Uj Letszam: ";
00675         cin.sync();
00676         cin » l;
00677         p->adat.setLetszam(l);
00678         DB.save();
00679         cout « "\n[+]Mentes..." « endl;
00680         return;
```

```
00681         break;
00682
00683       case 3:
00684         cout « "\n\t\t\t[?] Edzok hozzaadasa: ";
00685         cin.sync();
00686         cin » e;
00687         p->adat.addEdzo(e);
00688         DB.save();
00689         cout « "\n[+]Mentes..." « endl;
00690         return;
00691         break;
00692
00693       case 0:
00694         arg = 0;
00695         break;
00696
00697       default:
00698         cout « "\t\t\t[!] Nem letezo menu..." « endl;
00699         break;
00700     }
00701   }
00702 }
00703
00704 void Menu::ujMenu(Tipus T) {
00705   cout « endl;
00706   char nevbuffer[256];
00707   int letszam;
00708   long long int variszam;
00709   cout « "\n\t[?] Adja meg a csapat nevet: ";
00710   cin.sync();
00711   #ifdef __linux__
00712     cin.ignore(1, '\n');
00713   #endif
00714   #ifdef __APPLE__
00715     cin.ignore(1, '\n');
00716   #endif
00717   cin.getline(nevbuffer, 255);
00718   cout « "\t[?] Adja meg a csapat letszamat: ";
00719   cin.sync();
00720   cin » letszam;
00721   switch (T){
00722     case KEZI:
00723       cout « "\t[?] Adja meg a csapat tamogatasat: "; cin » variszam;
00724       cout « endl;
00725       DB.add(KEZI, nevbuffer, letszam, variszam);
00726       cout « "[+] Mentes...";
00727       DB.save();
00728       break;
00729
00730     case KOSAR:
00731       cout « "\t[?] Adja meg a csapat pompomlanyai szamat: "; cin » variszam;
00732       cout « endl;
00733       DB.add(KOSAR, nevbuffer, letszam, variszam);
00734       cout « "[+] Mentes...";
00735       DB.save();
00736       break;
00737
00738     case FOCI:
00739       cout « "\t[?] Adja meg a csapat edzoinek szamat: "; cin » variszam;
00740       cout « endl;
00741       DB.add(FOCI, nevbuffer, letszam, variszam);
00742       cout « "[+] Mentes...";
00743       DB.save();
00744       break;
00745
00746     case NINCS:
00747       break;
00748   }
00749 }
00750
00751 void Menu::foMenu() {
00752   cout « "\n\n\n\n[!] Sport Nyilvantarto Rendszer!\n\n\n" « endl;
00753
00754   int arg = -1;
00755   while (arg != 0) {
00756     cout « "\n\n[+] FOMENU" « endl;
00757     cout «
      "=============================================================================================================="
      « endl;
00758     cout « "| (1): Csapatok listazasa | (2): Kezilabda menu | (3): Kosarlabda menu | (4): Focilabda
      menu | (0): Kilepes |" « endl;
00759     cout «
      "=============================================================================================================="
      « endl;
00760     cout « "[!] Valasszon a listabol. Adja meg a menu szamat!" « endl;
00761     cout « "[?] Menu szama: ";
00762     cin.sync();
```

```
00763     cin » arg;
00764     switch (arg)
00765     {
00766     case 1:
00767       printAll();
00768       break;
00769     case 2:
00770       keziMenu();
00771       break;
00772     case 3:
00773       kosarMenu();
00774       break;
00775     case 4:
00776       fociMenu();
00777       break;
00778     case 0:
00779       break;
00780     default:
00781       cout « "\n[!] Nem letezo menu..." « endl;
00782       break;
00783     }
00784   }
00785   cout « "[+] Kilepes..." « endl;
00786 }
00787
```

5.13 menu.h

```
00001 #ifndef MENU_H
00002 #define MENU_H
00003
00004 #include "nyilvantartas.h"
00005 #include "memtrace.h"
00006
00008 class Menu {
00009   private:
00011     Nyilvantartas DB;
00012   public:
00013
00015     Menu();
00016
00018     ~Menu();
00019
00022     Nyilvantartas getNyilvantartas() const { return DB;}
00023
00025     void printAll() const;
00026
00028     void printKezi() const;
00029
00031     void printKosar() const;
00032
00034     void printFoci() const;
00035
00038     void printOne(KeziListaElem *) const;
00039
00042     void printOne(KosarListaElem *) const;
00043
00046     void printOne(FociListaElem *) const;
00047
00050     int maxStdRowLen() const;
00051
00055     int maxStdRowLen(Tipus) const;
00056
00060     int getStdRowLen(KeziListaElem *) const;
00061
00065     int getStdRowLen(KosarListaElem *) const;
00066
00070     int getStdRowLen(FociListaElem *) const;
00071
00074     void foMenu();
00075
00077     void keziMenu();
00078
00080     void kosarMenu();
00081
00083     void fociMenu();
00084
00086     void keresKeziMenu();
00087
00089     void keresKosarMenu();
00090
00092     void keresFociMenu();
00093
```

```
00096     void editKeziMenu(KeziListaElem *);
00097
00100     void editKosarMenu(KosarListaElem*);
00101
00104     void editFociMenu(FociListaElem *);
00105
00108     void ujMenu(Tipus);
00109 };
00110
00111 #endif
```

5.14 nyilvantartas.cpp

```
00001 #include <cstring>
00002 #include <cmath>
00003 #include <fstream>
00004 #include "csapat.h"
00005 #include "kezi.h"
00006 #include "kosar.h"
00007 #include "foci.h"
00008 #include "nyilvantartas.h"
00009 #include "memtrace.h"
00010
00011 using std::fstream;
00012 using std::ios;
00013 using std::getline;
00014
00015
00016 Nyilvantartas::~Nyilvantartas(){
00017   delKezi();
00018   delKosar();
00019   delFoci();
00020 }
00021
00022 KeziListaElem *Nyilvantartas::ujKezi() {
00023   // ha üres a lista
00024   if (keziCS == nullptr) {
00025     keziCS = new KeziListaElem[1];
00026     keziCS->kovi = nullptr;
00027     return keziCS;
00028   } /*egyébként ha nem üres a lista*/ else {
00029     KeziListaElem *i; for (i=keziCS;i->kovi!=nullptr;i=i->kovi){}
00030     i->kovi = new KeziListaElem[1];
00031     i->kovi->kovi = nullptr;
00032     return i->kovi;
00033   }
00034 }
00035
00036 void Nyilvantartas::addKezi(const char* csnev, const int letszam, const int tamogatas){
00037   KeziListaElem *editable = ujKezi();
00038   editable->adat.setNev(csnev);
00039   editable->adat.setLetszam(letszam);
00040   editable->adat.addTamogatas(tamogatas);
00041 }
00042
00043 KosarListaElem *Nyilvantartas::ujKosar() {
00044   // ha üres a lista
00045   if (kosarCS == nullptr) {
00046     kosarCS = new KosarListaElem[1];
00047     kosarCS->kovi = nullptr;
00048     return kosarCS;
00049   } /*egyébként ha nem üres a lista*/ else {
00050     KosarListaElem *i; for (i=kosarCS;i->kovi!=nullptr;i=i->kovi){}
00051     i->kovi = new KosarListaElem[1];
00052     i->kovi->kovi = nullptr;
00053     return i->kovi;
00054   }
00055 }
00056
00057 void Nyilvantartas::addKosar(const char* csnev, const int letszam, const int pompomn){
00058   KosarListaElem *editable = ujKosar();
00059   editable->adat.setNev(csnev);
00060   editable->adat.setLetszam(letszam);
00061   editable->adat.addPompom(pompomn);
00062 }
00063
00064 FociListaElem *Nyilvantartas::ujFoci() {
00065   // ha üres a lista
00066   if (fociCS == nullptr) {
00067     fociCS = new FociListaElem[1];
00068     fociCS->kovi = nullptr;
00069     return fociCS;
00070   } /*egyébként ha nem üres a lista*/ else {
```

```
00071     FociListaElem *i; for (i=fociCS;i->kovi!=nullptr;i=i->kovi){}
00072     i->kovi = new FociListaElem[1];
00073     i->kovi->kovi = nullptr;
00074     return i->kovi;
00075   }
00076 }
00077
00078 void Nyilvantartas::addFoci(const char* csnev, const int letszam, const int edzok){
00079   FociListaElem *editable = ujFoci();
00080   editable->adat.setNev(csnev);
00081   editable->adat.setLetszam(letszam);
00082   editable->adat.addEdzo(edzok);
00083 }
00084
00085 void Nyilvantartas::add(const Tipus T, const char* csnev, const int letszam, const int vari){
00086   if (T == KEZI) { addKezi(csnev, letszam, vari); }
00087   else if (T == KOSAR) { addKosar(csnev, letszam, vari); }
00088   else if (T == FOCI)  { addFoci(csnev, letszam, vari); }
00089 }
00090
00091 KeziListaElem   *Nyilvantartas::findKezi(const char *csapatnev) const {
00092   if (keziCS != nullptr) {
00093     KeziListaElem *i;
00094     for (i=keziCS; i != nullptr; i = i->kovi){
00095       if (i->adat == csapatnev) {
00096         return i;
00097       }
00098     }
00099     return nullptr;
00100   } else {
00101     return nullptr;
00102   }
00103 }
00104
00105 KosarListaElem  *Nyilvantartas::findKosar(const char *csapatnev) const {
00106   if (kosarCS != nullptr) {
00107     KosarListaElem *i;
00108     for (i=kosarCS; i != nullptr; i = i->kovi){
00109       if (i->adat == csapatnev) {
00110         return i;
00111       }
00112     }
00113     return nullptr;
00114   } else {
00115     return nullptr;
00116   }
00117 }
00118
00119 FociListaElem   *Nyilvantartas::findFoci(const char *csapatnev) const {
00120   if (fociCS != nullptr) {
00121     FociListaElem *i;
00122     for (i=fociCS; i != nullptr; i = i->kovi){
00123       if (i->adat == csapatnev) {
00124         return i;
00125       }
00126     }
00127     return nullptr;
00128   } else {
00129     return nullptr;
00130   }
00131 }
00132
00133 void Nyilvantartas::rm(KeziListaElem *& torlendo) {
00134   if (torlendo == nullptr) {return;}
00135   // ha az elejerol kell
00136   if (torlendo == keziCS) {
00137     KeziListaElem *ujeleje = keziCS->kovi;
00138     delete[] torlendo;
00139     keziCS = ujeleje;
00140   } else /* Ha a kozeperol v a vegerol torlunk */ {
00141     KeziListaElem *i; for (i = keziCS; i->kovi != torlendo; i = i->kovi) {}
00142     i->kovi = i->kovi->kovi;
00143     delete[] torlendo;
00144   }
00145 }
00146
00147 void Nyilvantartas::rm(KosarListaElem *& torlendo) {
00148   if (torlendo == nullptr) {return;}
00149   // ha az elejerol kell
00150   if (torlendo == kosarCS) {
00151     KosarListaElem *ujeleje = kosarCS->kovi;
00152     delete[] torlendo;
00153     kosarCS = ujeleje;
00154   } else /* Ha a kozeperol v a vegerol torlunk */ {
00155     KosarListaElem *i; for (i = kosarCS; i->kovi != torlendo; i = i->kovi) {}
00156     i->kovi = i->kovi->kovi;
00157     delete[] torlendo;
```

```
00158   }
00159 }
00160
00161 void Nyilvantartas::rm(FociListaElem *& torlendo) {
00162   if (torlendo == nullptr) {return;}
00163   // ha az elejerol kell
00164   if (torlendo == fociCS) {
00165     FociListaElem *ujeleje = fociCS->kovi;
00166     delete[] torlendo;
00167     fociCS = ujeleje;
00168   } else /* Ha a kozeperol v a vegerol torlunk */ {
00169     FociListaElem *i; for (i = fociCS; i->kovi != torlendo; i = i->kovi) {}
00170     i->kovi = i->kovi->kovi;
00171     delete[] torlendo;
00172   }
00173 }
00174
00175 KeziListaElem   *Nyilvantartas::getKeziLista() const {return keziCS;}
00176
00177 KosarListaElem  *Nyilvantartas::getKosarLista() const {return kosarCS;}
00178
00179 FociListaElem   *Nyilvantartas::getFociLista() const {return fociCS;}
00180
00181 bool Nyilvantartas::loadKezi() {
00182   // File leírás:
00183   // Csapatnév [TABULÁTO] Csapatlétszám [TABULÁTOR] Támogatás
00184
00185   fstream f;
00186   f.open("kezi.txt", ios::in);
00187   if (f.is_open()) {
00188     char c;
00189     char *csNevSTR = nullptr;
00190     char *letszamSTR = nullptr;
00191     char *variszamSTR = nullptr;
00192     int tabulatorCount = 0;
00193     int letszam = 0;
00194     int variszam = 0;
00195
00196     // A filet végignyálazza betűnként.
00197     while (f.get(c) && c != EOF) {
00198       //Ha sor vége van, új lista láncolása logika és reset.
00199       if (c == '\n') {
00200         tabulatorCount = 0;
00201         sscanf(letszamSTR, "%d", &letszam);
00202         sscanf(variszamSTR, "%d", &variszam);
00203         add(KEZI, csNevSTR, letszam, variszam);
00204         strdel(csNevSTR); strdel(letszamSTR); strdel(variszamSTR);
00205
00206       } else if (c == '\t') {tabulatorCount++;} // ha tab, más adadtba pakolunk bele betűket.
00207       else {
00208         // a megfelelő adatba a betűk belepakolása.
00209         if (tabulatorCount==0) {straddc(csNevSTR, c);}
00210         else if (tabulatorCount==1) {straddc(letszamSTR, c);}
00211         else if (tabulatorCount==2) {straddc(variszamSTR, c);}
00212       }
00213     }
00214     #ifdef ROSSZAVEGE
00215     //az eof után az utsó sor még nincs lerendezve
00216     sscanf(letszamSTR, "%d", &letszam);
00217     sscanf(variszamSTR, "%d", &variszam);
00218     add(KEZI, csNevSTR, letszam, variszam);
00219     strdel(csNevSTR); strdel(letszamSTR); strdel(variszamSTR);
00220     #endif
00221     f.close();
00222     return true;
00223   } else {
00224     return false;
00225   }
00226 }
00227
00228 bool Nyilvantartas::loadKosar() {
00230   // Csapatnév [TABULÁTO] Csapatlétszám [TABULÁTOR] Pompomlányokszáma
00231
00232   fstream f;
00233   f.open("kosar.txt", ios::in);
00234   if (f.is_open()) {
00235     char c;
00236     char *csNevSTR = nullptr;
00237     char *letszamSTR = nullptr;
00238     char *variszamSTR = nullptr;
00239     int tabulatorCount = 0;
00240     int letszam = 0;
00241     int variszam = 0;
00242
00243     // A filet végignyálazza betűnként.
00244     while (f.get(c) && c != EOF) {
00245       //Ha sor vége van, új lista láncolása logika és reset.
```

```
00246       if (c == '\n') {
00247         tabulatorCount = 0;
00248         sscanf(letszamSTR, "%d", &letszam);
00249         sscanf(variszamSTR, "%d", &variszam);
00250         add(KOSAR, csNevSTR, letszam, variszam);
00251         strdel(csNevSTR); strdel(letszamSTR); strdel(variszamSTR);
00252 
00253       } else if (c == '\t') {tabulatorCount++;} // ha tab, más adadtba pakolunk bele betűket.
00254       else {
00255         // a megfelelő adatba a betűk belepakolása.
00256         if (tabulatorCount==0) {straddc(csNevSTR, c);}
00257         else if (tabulatorCount==1) {straddc(letszamSTR, c);}
00258         else if (tabulatorCount==2) {straddc(variszamSTR, c);}
00259       }
00260     }
00261     #ifdef ROSSZAVEGE
00262     //az eof után az utsó sor még nincs lerendezve
00263     sscanf(letszamSTR, "%d", &letszam);
00264     sscanf(variszamSTR, "%d", &variszam);
00265     add(KOSAR, csNevSTR, letszam, variszam);
00266     strdel(csNevSTR); strdel(letszamSTR); strdel(variszamSTR);
00267     #endif
00268     f.close();
00269     return true;
00270   } else {
00271     return false;
00272   }
00273 }
00274 
00275 bool Nyilvantartas::loadFoci() {
00276   // File leírás:
00277   // Csapatnév [TABULÁTO] Csapatlétszám [TABULÁTOR] Edzokszama
00278 
00279   fstream f;
00280   f.open("foci.txt", ios::in);
00281   if (f.is_open()) {
00282     char c;
00283     char *csNevSTR = nullptr;
00284     char *letszamSTR = nullptr;
00285     char *variszamSTR = nullptr;
00286     int tabulatorCount = 0;
00287     int letszam = 0;
00288     int variszam = 0;
00289 
00290     // A filet végignyálazza betűnként.
00291     while (f.get(c) && c != EOF) {
00292       //Ha sor vége van, új lista láncolása logika és reset.
00293       if (c == '\n') {
00294         tabulatorCount = 0;
00295         sscanf(letszamSTR, "%d", &letszam);
00296         sscanf(variszamSTR, "%d", &variszam);
00297         add(FOCI, csNevSTR, letszam, variszam);
00298         strdel(csNevSTR); strdel(letszamSTR); strdel(variszamSTR);
00299 
00300       } else if (c == '\t') {tabulatorCount++;} // ha tab, más adadtba pakolunk bele betűket.
00301       else {
00302         // a megfelelő adatba a betűk belepakolása.
00303         if (tabulatorCount==0) {straddc(csNevSTR, c);}
00304         else if (tabulatorCount==1) {straddc(letszamSTR, c);}
00305         else if (tabulatorCount==2) {straddc(variszamSTR, c);}
00306       }
00307     }
00308     #ifdef ROSSZAVEGE
00309     //az eof után az utsó sor még nincs lerendezve
00310     sscanf(letszamSTR, "%d", &letszam);
00311     sscanf(variszamSTR, "%d", &variszam);
00312     add(FOCI, csNevSTR, letszam, variszam);
00313     strdel(csNevSTR); strdel(letszamSTR); strdel(variszamSTR);
00314     #endif
00315     f.close();
00316     return true;
00317   } else {
00318     return false;
00319   }
00320 }
00321 
00322 bool Nyilvantartas::load() {
00323   if (!loadKezi())  {return false;}
00324   if (!loadKosar()) {return false;}
00325   if (!loadFoci())  {return false;}
00326   return true;
00327 }
00328 
00329 int Nyilvantartas::intlen(const long long int szam) {
00330   for (int i = 0; i <= 64; i++) {
00331     if (szam % (long long int) pow(10, i) == szam) {
00332       return i==0 ? 1 : i;
```

```
00333     }
00334   }
00335   return 0;
00336 }
00337
00338 void Nyilvantartas::straddc(char *&str, const char c){
00339   if (str == nullptr) {
00340     str = new char[2];
00341     str[0] = c;
00342     str[1] = '\0';
00343   } else {
00344     int l = strlen(str);
00345     char *temp = new char[l+1];
00346     strcpy(temp, str);
00347     delete[] str;
00348     str = new char[l+2];
00349     strcpy(str, temp);
00350     str[l] = c; str[l+1] = '\0';
00351     delete[] temp;
00352   }
00353 }
00354
00355 void Nyilvantartas::strdel(char *&str){
00356   if (str == nullptr) {return;}
00357   delete[] str; str = nullptr;
00358 }
00359
00360 void Nyilvantartas::saveKezi() const {
00361   fstream f;
00362   f.open("kezi.txt", ios::out);
00363   if (!f.is_open()) {return;}
00364   for (KeziListaElem *i = keziCS; i != nullptr; i = i->kovi) {
00365     f « i->adat.getNev() « "\t" « i->adat.getLetszam() «
00366     "\t" « i->adat.getTamogatas() « "\n";
00367   }
00368   f.close();
00369 }
00370
00371 void Nyilvantartas::saveKosar() const {
00372   fstream f;
00373   f.open("kosar.txt", ios::out);
00374   if (!f.is_open()) {return;}
00375   for (KosarListaElem *i = kosarCS; i != nullptr; i = i->kovi) {
00376     f « i->adat.getNev() « "\t" « i->adat.getLetszam() «
00377     "\t" « i->adat.getPomPomDb() « "\n";
00378   }
00379   f.close();
00380 }
00381
00382 void Nyilvantartas::saveFoci() const {
00383   fstream f;
00384   f.open("foci.txt", ios::out);
00385   if (!f.is_open()) {return;}
00386   for (FociListaElem *i = fociCS; i != nullptr; i = i->kovi) {
00387     f « i->adat.getNev() « "\t" « i->adat.getLetszam() «
00388     "\t" « i->adat.getEdzokszama() « "\n";
00389   }
00390   f.close();
00391 }
00392
00393 void Nyilvantartas::save() const {
00394   saveKezi();
00395   saveKosar();
00396   saveFoci();
00397 }
```

5.15 nyilvantartas.h

```
00001 #ifndef NYILVANTARTAS_H
00002 #define NYILVANTARTAS_H
00003
00004 #include "csapat.h"
00005 #include "kezi.h"
00006 #include "kosar.h"
00007 #include "foci.h"
00008 #include "memtrace.h"
00009
00011 struct KeziListaElem {
00013   Kezi adat;
00014
00016   KeziListaElem *kovi;
00017 };
00018
```

```
00020 struct KosarListaElem {
00022   Kosar adat;
00023
00025   KosarListaElem *kovi;
00026 };
00027
00029 struct FociListaElem {
00031   Foci adat;
00032
00034   FociListaElem *kovi;
00035 };
00036
00040 class Nyilvantartas {
00041   private:
00043     KeziListaElem   *keziCS;
00044
00046     KosarListaElem  *kosarCS;
00047
00049     FociListaElem   *fociCS;
00050
00052     void delKezi(){
00053       if (keziCS != nullptr) {
00054         KeziListaElem* i = keziCS;
00055         while (i != nullptr) {
00056           KeziListaElem *kov = i->kovi;
00057           delete[] i;
00058           i = kov;
00059         }
00060       }
00061     }
00062
00064     void delKosar(){
00065       if (kosarCS != nullptr) {
00066         KosarListaElem* i = kosarCS;
00067         while (i != nullptr) {
00068           KosarListaElem *kov = i->kovi;
00069           delete[] i;
00070           i = kov;
00071         }
00072       }
00073     }
00074
00076     void delFoci(){
00077       if (fociCS != nullptr) {
00078         FociListaElem* i = fociCS;
00079         while (i != nullptr) {
00080           FociListaElem *kov = i->kovi;
00081           delete[] i;
00082           i = kov;
00083         }
00084       }
00085     }
00086
00087   public:
00088
00094     static int intlen(const long long int);
00095
00099     static void straddc(char *&, const char);
00100
00103     static void strdel(char *&);
00104
00107     Nyilvantartas() : keziCS(nullptr), kosarCS(nullptr), fociCS(nullptr) {}
00108
00110     ~Nyilvantartas();
00111
00115     KeziListaElem   *ujKezi();
00116
00120     KosarListaElem  *ujKosar();
00121
00125     FociListaElem   *ujFoci();
00126
00131     void addKezi(const char*, const int, const int);
00132
00137     void addKosar(const char*, const int, const int);
00138
00143     void addFoci(const char*, const int, const int);
00144
00150     void add(const Tipus, const char*, const int, const int);
00151
00155     KeziListaElem   *findKezi(const char*) const;
00156
00160     KosarListaElem  *findKosar(const char*) const;
00161
00165     FociListaElem   *findFoci(const char*) const;
00166
00169     void rm(KeziListaElem *&);
00170
```

```
00173     void rm(KosarListaElem *&);
00174
00177     void rm(FociListaElem *&);
00178
00181     KeziListaElem   *getKeziLista() const;
00182
00185     KosarListaElem  *getKosarLista() const;
00186
00189     FociListaElem   *getFociLista() const;
00190
00193     bool loadKezi();
00194
00197     bool loadKosar();
00198
00201     bool loadFoci();
00202
00204     void saveKezi() const;
00205
00207     void saveKosar() const;
00208
00210     void saveFoci() const;
00211
00215     bool load();
00216
00219     void save() const;
00220 };
00221
00222 #endif
```

Index