HÁZI FELADAT

Programozás alapjai 2.
Kész

Kerekes Adrienne
GFVHSO
2022.05.15.

Tartalomjegyzék

Kerekes Adrienne
GFVHSO

1. A program célja

A program egy nyilvántartó program alapvető funkcióit látja el. Képes tárolni különböző állatok adatait és a hozzájuk tartozó megjegyzéseket. A program lehetőséget ad új bejegyzések létrehozására, a már meglévők módosítására, törlésére. Felhasználói felület nem kerül kialakításra.

2. Pontosított feladatspecifikáció

A program alapvetően képes kutyákról, macskákról és papagájokról adatokat tárolni, módosítani, törölni. (Későbbiekben a megrendelő kívánságainak megfelelően ezek a lehetőségek bővíthetőek a kódban történő egyszerűbb módosításokkal.)

Az adatok külön-külön egy-egy iterátoros generikus listában vannak tárolva a típusuknak megfelelően. A lista alapvető funkciói: beszúrás, módosítás, törlés, beolvasás, kiírás, az egész lista törlése, címképzés, illetve az iterátorhoz kapcsolódó elengedhetetlen függvények például pointer összehasonlítás, pointer növelése stb. Hibás indexelés esetén kivételt dob a program.

Az adatok tárolása forrásfájlban is hasonló. Minden típus adatai külön adatfájlba kerülnek és onnan kerülnek beolvasásra.

Az adatok megjelenítése a standard outputon történik.

3. Objektum terv

A generikus lista egy sablonnal van megvalósítva, ami paraméterként átveszi a lista elemeinek típusát. Alaposztályok String, Datum, Tulaj, Allat. Származtatott osztályok Kutya, Macska, Papagaj, amik az Allat osztályból vannak származtatva.

4. Megvalósítás

A feladat megoldása során egy osztálysablon és a felhasznált osztályok lettek elkészítve. A sablon felhasználja <iostream> <fstream> <stdexcept> <string> <sstream> osztályokat. Az osztályok végleges interfésze a függvények számában változott. Az elkészült függvényeket és metódusokat a további fejezetekben mutatom be. A tesztelés egy kiegészítő fájlal lett megoldva: gtest_lite.h.

4.2. Az elkészített osztálysablon bemutatása

Osztályok

class	**iterator**
struct	**ListaElem**

Publikus tagfüggvények

	Lista () Strázsa létrehozása. Részletek...
void	**beszur** (const T &dat)
void	**torol** (const T &dat)
void	**modosit** (const T &dat, const T &mod)
void	**beolvas** (const char *p)
iterator	**begin** () Létrehoz egy iterátort és az elejére állítja. Részletek...
iterator	**end** () Létrehozza és az utolsó elem után állítja. Részletek...
	~Lista () Destruktor. Részletek...

Privát tagfüggvények

Lista &	**operator=** (const **Lista** &)

Privát attribútumok

ListaElem *	**elso**
ListaElem *	**akt**

Barátok

std::ostream &	**operator<<** (std::ostream &os, const **Lista** &rhs)
std::ofstream &	**operator<<** (std::ofstream &of, const **Lista** &rhs)

4.2.1. Iterátor osztály

Szerepe: általánosan kezelhetjük a tárolókat, azok belső megvalósításának ismerete nélkül. Legfontosabb műveletei:

- éppen aktuális elem elérése (operator*, operator->),
- következő elemre lépés (operator++),
- mutatók összehasonlítása (operator==, operator!=),
- mutatóobjektum létrehozása az első elemre (begin()),
- mutatóobjektum létrehozása az utolsó utáni elemre (end()).

Publikus tagfüggvények

	iterator ()
	iterator (const **Lista** &I)
iterator &	**operator++** ()
iterator	**operator++** (int)
bool	**operator!=** (const **iterator** &i) const
T &	**operator*** ()
T *	**operator->** ()

Privát attribútumok

ListaElem *	**akt**

Konstruktorok és destruktorok dokumentációja

◆ iterator() [1/2]

template<class T >

Lista< T >::iterator::iterator ( ) inline

Iterátor konstruktor Végére állítja az iterátort

Definíció a(z) **list.hpp** fájl **104**. sorában.

◆ iterator() [2/2]

template<class T >

Lista< T >::iterator::iterator (const **Lista** & I) inline

Iterátor konstruktor Elejére állítja az iterátort

Paraméterek

I - lista refernciája

Definíció a(z) **list.hpp** fájl **109**. sorában.

Adattagok dokumentációja

◆ akt

template<class T >

ListaElem* **Lista**< T >::iterator::akt private

Definíció a(z) **list.hpp** fájl **99**. sorában.

Tagfüggvények dokumentációja

◆ operator!=()

template<class T >

bool **Lista**< T >::iterator::operator!= (const **iterator** & i) const inline

Összehasonlít

Paraméterek

i - iterátor referencia

Visszatérési érték

bool

Definíció a(z) **list.hpp** fájl **136**. sorában.

◆ operator*()

template<class T >

T & **Lista**< T >::iterator::operator* ( ) inline

Indirekció

Visszatérési érték

osztály referencia

Definíció a(z) **list.hpp** fájl **142**. sorában.

◆ operator++() [1/2]

template<class T >

iterator & **Lista**< T >::iterator::operator++ ( ) inline

Növeli az iterátort (pre)

Visszatérési érték

iterátor referencia

Definíció a(z) **list.hpp** fájl **116**. sorában.

◆ operator++() [2/2]

template<class T >

iterator Lista< T >::iterator::operator++ (int) inline

Növeli az iterátort (post)

Visszatérési érték

iterátor referencia

Definíció a(z) **list.hpp** fájl **127**. sorában.

◆ operator->()

template<class T >

T * **Lista**< T >::iterator::operator-> ( ) inline

Indirekció

Visszatérési érték

osztály pointer

Definíció a(z) **list.hpp** fájl **149**. sorában.

4.2.2. Publikus tagfüggvények

Konstruktorok és destruktorok dokumentációja

◆ Lista()

template<class T >

Lista< T >::**Lista** ( ) inline

Strázsa létrehozása.

Definíció a(z) **list.hpp** fájl **37**. sorában.

◆ ~Lista()

template<class T >

Lista< T >::~**Lista**

Destruktor.

Definíció a(z) **list.hpp** fájl **161**. sorában.

Tagfüggvények dokumentációja

◆ beszur()

template<class T >

void **Lista**< T >::beszur (const T & dat)

Elem beszúrása listába

Paraméterek

dat - beszúrandó adat refernciája

Definíció a(z) **list.hpp** fájl **170**. sorában.

A paraméterként kapott referencia alapján először megnézi, hogy létezik-e ugyanilyen elem a listában. Ha nem, akkor beszúrja a lista végére a paraméterként kapott elemet. Ha létezik már olyan elem a listában, mint a beszúrandó elem, akkor hibát dob, illetve értesíti a felhasználót.

A paraméterként kapott referencia alapján először megnézi, hogy létezik-e ugyanilyen elem a listában. Ha nem, akkor értesíti a felhasználót a sikertelen törlésről. Ha létezik olyan elem a listában, mint a törlendő elem, akkor törli az elemet és értesíti erről a felhasználót.

◆ torol()

```
template<class T >
void Lista< T >::torol ( const T &  dat )
```

Elem törlése listából

Paraméterek

dat - törlendő adat refernciája

Definíció a(z) **list.hpp** fájl **190**. sorában.

◆ modosit()

```
template<class T >
void Lista< T >::modosit ( const T &  dat,
                           const T &  mod
                         )
```

Elem módosítása a listában

Paraméterek

dat - módosítandó adat refernciája

mod - módosító adat refernciája

Definíció a(z) **list.hpp** fájl **217**. sorában.

A paraméterként kapott dat nevű referencia alapján először megnézi, hogy létezik-e ugyanilyen elem a listában. Ha nem, akkor értesíti a felhasználót a sikertelen módosításról. Ha létezik olyan elem a listában, mint a módosítandó dat nevű elem, akkor módosítja az elemet a mod nevű paraméterre és értesíti erről a felhasználót.

◆ beolvas()

```
template<class T >
void Lista< T >::beolvas ( const char *  p )
```

Elemek beolvasása adatfájlból

Paraméterek

p - forrásfájl neve "nev.txt" formátumban

Definíció a(z) **list.hpp** fájl **233**. sorában.

A függvény a paraméterként kapott fájlnév alapján megnyitja az adatfájlt, majd beolvas a fájlból egy sort, azt tovább bontja „;” mentén, létrehozza a megfelelő objektumot és beszúrja a listába. Ha rossz fájlnevet adnak meg paraméterként, akkor hibát dob és értesíti a felhasználót erről.

◆ begin()

template<class T >

iterator Lista< T >::begin ( ) inline

Létrehoz egy iterátort és az elejére állítja.

Definíció a(z) **list.hpp** fájl **89**. sorában.

◆ end()

template<class T >

iterator Lista< T >::end ( ) inline

Létrehozza és az utolsó elem után állítja.

Definíció a(z) **list.hpp** fájl **94**. sorában.

Adattagok dokumentációja

◆ akt

template<class T >

ListaElem* **Lista**< T >::akt private

Definíció a(z) **list.hpp** fájl **31**. sorában.

◆ elso

template<class T >

ListaElem* **Lista**< T >::elso private

Definíció a(z) **list.hpp** fájl **30**. sorában.

Barát és kapcsolódó függvények dokumentációja

◆ operator<< [1/2]

```
template<class T >
std::ofstream & operator<< ( std::ofstream &       of,
                             const Lista< T > &  rhs
                           )                                  friend
```

Globális fájlinserter

Paraméterek

of - output stream referencia

rhs - **Lista** referencia

Visszatérési érték

output stream referencia

Definíció a(z) **list.hpp** fájl **59**. sorában.

◆ operator<< [2/2]

```
template<class T >
std::ostream & operator<< ( std::ostream &       os,
                            const Lista< T > &  rhs
                          )                                   friend
```

Globális inserter

Paraméterek

os - output stream referencia

rhs - **Lista** referencia

Visszatérési érték

output stream referencia

Definíció a(z) **list.hpp** fájl **43**. sorában.

Azért kellett friend operátort csinálnom, mert csak így tudtam használni az osztályok globális insertereit.

4.2.3. ListaElem struktúra

Publikus tagfüggvények

	ListaElem (**ListaElem** *p=NULL) Konstruktor. Részletek...
ListaElem &	**operator=** (const **ListaElem**)

Publikus attribútumok

T	**adat**
ListaElem *	**kov**

Konstruktorok és destruktorok dokumentációja

◆ ListaElem()

template<class T >

Lista< T >::ListaElem::ListaElem (**ListaElem** * p = NULL) inline

Konstruktor.

Definíció a(z) **list.hpp** fájl **25**. sorában.

Tagfüggvények dokumentációja

◆ operator=()

template<class T >

ListaElem & **Lista**< T >::ListaElem::operator= (const ListaElem)

Adattagok dokumentációja

◆ adat

template<class T >

T **Lista**< T >::ListaElem::adat

Definíció a(z) **list.hpp** fájl **22**. sorában.

◆ kov

template<class T >

ListaElem* **Lista**< T >::ListaElem::kov

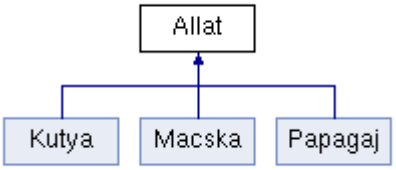
Definíció a(z) **list.hpp** fájl **23**. sorában.

4.3. Az elkészített osztályok bemutatása

Ebben a fejezetben a következő osztályokról és tagfüggvényeikről, illetve operátoraikról lesz szó: Allat, Datum, String, Kutya, Macska, Papagaj.

Ezek az osztályok a <iostream> <fstream> <iomanip> <cstring> <sstream> <ctime> osztályokat és a "memtrace.h" fájlt használják.

4.3.1. Állat osztály



Ebből az osztályból lett származtatva A Kutya, Macska és Papagaj osztályok. Adattagjai a String osztály, - ami egy saját készítésű string kezelő osztály - egy-egy példánya.

Publikus tagfüggvények

	Allat () Default konstruktor. Részletek...
	Allat (const char ***faj**, const char ***nev**, const **Datum** &**szul**)
String	**getFaj** () const
void	**setFaj** (const char *p)
String	**getNev** () const
void	**setNev** (const char *p)
Datum	**getSzul** () const
void	**setSzul** (**Datum** &d)
virtual	**~Allat** () Virtuális destruktor. Részletek...

Védett attribútumok

String	**faj**
String	**nev**
Datum	**szul**

Az osztálynak két féle konstruktora van. Egy alap paraméter nélkül hívható konstruktor, illetve egy paraméteres konstruktor.

A paraméter nélkül létrehozott példányban minden adattag egy üres string. A paraméteresen hívott konstruktorban kötelező minden adattagnak értéket adni. Ha valahova üres adattartalmú string kerül, akkor hibát dob a konstruktor.

Az osztály destruktora virtuális destruktor, mivel származtatva van belőle másik osztály.

Konstruktorok és destruktorok dokumentációja

◆ Allat() [1/2]

Allat::Allat ( ) inline

Default konstruktor.

Definíció a(z) **allat.h** fájl **20**. sorában.

◆ Allat() [2/2]

```
Allat::Allat ( const char *      faj,
               const char *      nev,
               const Datum &     szul
             )
```
inline

Konstruktor beállítja az attribútumokat

Paraméterek

faj - faj megnevezése

nev - név megadása

szul - születési év megadása

Definíció a(z) **allat.h** fájl **26**. sorában.

◆ ~Allat()

virtual Allat::~Allat ( ) inline virtual

Virtuális destruktor.

Definíció a(z) **allat.h** fájl **67**. sorában.

Tagfüggvények dokumentációja

◆ getFaj()

String Allat::getFaj ( ) const inline

Faj lekérdezése

Visszatérési érték
- faj

Definíció a(z) **allat.h** fájl **36**. sorában.

◆ getNev()

String Allat::getNev ( ) const inline

Név lekérdezése

Visszatérési érték
- név

Definíció a(z) **allat.h** fájl **48**. sorában.

◆ getSzul()

Datum Allat::getSzul ( ) const inline

Születési dátum lekérdezése

Visszatérési érték
- dátum

Definíció a(z) **allat.h** fájl **60**. sorában.

A fentebb lévő függvények lekérdezik az osztály privát adattagjainak értékét és konstans String, illetve Datum típusú értékkel térnek vissza.

A következő függvények az osztály privát adattagjainak adnak értéket. Bemeneti paraméterként egy karaktersorozat pointerét kapják. Itt szintén megvizsgálásra kerül, hogy a paraméter egy értékkel nem rendelkező adat-e. Ha igen, akkor hibát dob a függvény.

◆ setFaj()

void Allat::setFaj (const char * p) inline

Faj beállítása

Paraméterek

p - karaktersorozat

Definíció a(z) **allat.h** fájl **40**. sorában.

◆ setNev()

void Allat::setNev (const char * p) inline

Nev beállítása

Paraméterek

p - karaktersorozat

Definíció a(z) **allat.h** fájl **52**. sorában.

A setSzul nevű függvény paraméterként egy Datum referenciát kap. Ha nem megfelelő a dátum értéke, azt a Datum osztály már a Datum típusú adat létrehozásánál lekezeli, tehát nem megfelelő adat itt nem kerülhet beállításra.

◆ setSzul()

void Allat::setSzul (**Datum** & d) inline

Szuletési dátum beállítása

Paraméterek

d - dátum referencia

Definíció a(z) **allat.h** fájl **64**. sorában.

Adattagok dokumentációja

◆ faj

String Allat::faj — protected

Definíció a(z) **allat.h** fájl **15**. sorában.

◆ nev

String Allat::nev — protected

Definíció a(z) **allat.h** fájl **16**. sorában.

◆ szul

Datum Allat::szul — protected

Definíció a(z) **allat.h** fájl **17**. sorában.

4.3.2. Tulaj osztály

Publikus tagfüggvények

	Tulaj () Default konstruktor. Részletek...
	Tulaj (const char *nev, const char *cim, const char *tel)
bool	**operator==** (const **Tulaj** &t) const
bool	**operator!=** (const **Tulaj** &t) const
String	**getNev** () const
void	**settNev** (const char *p)
String	**getCim** () const
void	**setCim** (const char *p)
String	**getTel** () const
void	**setTel** (const char *p)
	~Tulaj () Destruktor. Részletek...

Privát attribútumok

String	**tNev**
String	**cim**
String	**tel**

A tulaj osztály az állat osztályból származtatott osztályok egyik adattagja. Az osztály két féle konstruktorral rendelkezik. A paraméter nélküli konstruktor egy üres adattagokból álló példányt hoz létre. A paraméteres konstruktor használatakor kötelező az összes adattagot inicializálni. Üres adat megadása esetén hibát dob.

Konstruktorok és destruktorok dokumentációja

◆ Tulaj() [1/2]

Tulaj::Tulaj ( ) inline

Default konstruktor.

Definíció a(z) **tulaj.h** fájl **19**. sorában.

◆ Tulaj() [2/2]

```
Tulaj::Tulaj ( const char *  nev,
               const char *  cim,
               const char *  tel
             )
```

Konstruktor beállítja az attribútumokat

Paraméterek

nev - név megadása

cim - cím megadása

tel - telefonszám megadása

Definíció a(z) **tulaj.cpp** fájl **12**. sorában.

Tagfüggvények dokumentációja

◆ getCim()

String Tulaj::getCim ( ) const inline

Cím lekérdezése

Visszatérési érték

- cím

Definíció a(z) **tulaj.h** fájl **55**. sorában.

◆ getNev()

String Tulaj::getNev ( ) const inline

Név lekérdezése

Visszatérési érték

- név

Definíció a(z) **tulaj.h** fájl **43**. sorában.

◆ getTel()

String Tulaj::getTel ( ) const inline

Telefonszám lekérdezése

Visszatérési érték

- telefonszám

Definíció a(z) **tulaj.h** fájl **67**. sorában.

A getNev, getCim és getTel nevű függvények visszaadják az adattagokat konstans String típusként.

◆ operator!=()

bool Tulaj::operator!= (const **Tulaj** & t) const inline

Két tulaj nem egyezőségét vizsgálja

Paraméterek

d - jobb oldali operandus

Visszatérési érték

true, ha nem egyezik a két tulaj

Definíció a(z) **tulaj.h** fájl **37**. sorában.

◆ operator==()

bool Tulaj::operator== (const **Tulaj** & t) const — inline

Két tulaj egyezőségét vizsgálja

Paraméterek

d - jobb oldali operandus

Visszatérési érték

true, ha egyezik a két tulaj

Definíció a(z) **tulaj.h** fájl **30**. sorában.

Az összehasonlító operátorok adattag szinten hasonlítják össze a két példányt.

A setter függvények paraméterként megkapott karaktersorozat értékét beállítják a tagváltozóknak. Ha a paraméter üres adattartalmú, akkor hibát dobnak a függvények.

◆ setCim()

void Tulaj::setCim (const char * p) — inline

Cim beállítása

Paraméterek

p - karaktersorozat

Definíció a(z) **tulaj.h** fájl **59**. sorában.

◆ setTel()

void Tulaj::setTel (const char * p) — inline

Telefonszám beállítása

Paraméterek

p - karaktersorozat

Definíció a(z) **tulaj.h** fájl **71**. sorában.

◆ settNev()

void Tulaj::settNev (const char * p) — inline

Nev beállítása

Paraméterek

p - karaktersorozat

Definíció a(z) **tulaj.h** fájl **47**. sorában.

Adattagok dokumentációja

◆ cim

String Tulaj::cim — private

Definíció a(z) **tulaj.h** fájl **15**. sorában.

◆ tel

String Tulaj::tel — private

Definíció a(z) **tulaj.h** fájl **16**. sorában.

◆ tNev

String Tulaj::tNev — private

Definíció a(z) **tulaj.h** fájl **14**. sorában.

4.3.3. Dátum osztály

Publikus tagfüggvények

	Datum ()
	Datum (int **ev**, int **ho**, int **nap**)
int	**getEv** () const
void	**setEv** (int **ev**)
int	**getHo** () const
void	**setHo** (int **ho**)
int	**getNap** () const
void	**setNap** (int **nap**)
bool	**operator==** (const **Datum** &d) const
bool	**operator!=** (const **Datum** &d) const

Privát attribútumok

int	**ev**
int	**ho**
int	**nap**

Az állat osztály egyik adattagja.

Két féle konstruktora van. A paraméter nélkül hívható konstruktor az aznapi dátumot állítja be. A paraméteres konstruktor minden adattagja kötelezően megadandó. Ellenőrzi, hogy a hónapok száma 1 és 12 között legyen, a napok száma 1 és 31 között, illetve az év nem lehet több a jelenlegi évnél és nem lehet kevesebb mint 2000. Ezeknek a feltételeknek a megszegése esetén hibát dob.

Konstruktorok és destruktorok dokumentációja

◆ Datum() [1/2]

Datum::Datum ( )

Paraméter nélkül hívható konstruktor. Mai napot állítja be

Definíció a(z) **datum.cpp** fájl **19**. sorában.

◆ Datum() [2/2]

```
Datum::Datum ( int  ev,
               int  ho,
               int  nap
             )
```

Adott napot beállító konstruktor

Paraméterek

ev - év
hó - hónap
nap - nap

Definíció a(z) **datum.cpp** fájl **29**. sorában.

A dátum osztály getterei a beállított dátum év, hónap vagy nap adattagját adják vissza, aminek az értéke egy konstans egész típus.

Tagfüggvények dokumentációja

◆ getEv()

int Datum::getEv ( ) const inline

Év lekérdezése

Visszatérési érték

ev

Definíció a(z) **datum.h** fájl **33**. sorában.

◆ getHo()

int Datum::getHo ( ) const inline

Hónap lekérdezése

Visszatérési érték

honap

Definíció a(z) **datum.h** fájl **41**. sorában.

◆ getNap()

int Datum::getNap ( ) const inline

Nap lekérdezése

Visszatérési érték

nap

Definíció a(z) **datum.h** fájl **52**. sorában.

Az összehasonlító operátorok két dátum egyezőségét vagy nem egyezőségét vizsgálják adattag szinten. Visszatérési értékük logikai típusú igaz vagy hamis.

◆ operator!=()

bool Datum::operator!= (const **Datum** & d) const inline

Két dátum nem egyezőségét vizsgálja

Paraméterek

d - jobb oldali operandus

Visszatérési érték

true, ha nem egyezik a két dátum

Definíció a(z) **datum.h** fájl **71**. sorában.

◆ operator==()

bool Datum::operator== (const **Datum** & d) const inline

Két dátum egyezőségét vizsgálja

Paraméterek

d - jobb oldali operandus

Visszatérési érték

true, ha egyezik a két dátum

Definíció a(z) **datum.h** fájl **64**. sorában.

A setterek egy paraméterként megkapott egész értékre állítják be az osztály adattagjait. Itt is hasonló a hibakezelés, mint a konstruktor esetében.

◆ setEv()

void Datum::setEv (int ev)

Év beállítása

Paraméterek

- ev

Definíció a(z) **datum.cpp** fájl **42**. sorában.

◆ setHo()

void Datum::setHo (int ho) inline

Hónap beállítása

Paraméterek

- honap

Definíció a(z) **datum.h** fájl **45**. sorában.

◆ setNap()

void Datum::setNap (int nap) inline

Nap beállítása

Paraméterek

- nap

Definíció a(z) **datum.h** fájl **56**. sorában.

Adattagok dokumentációja

◆ ev

int Datum::ev private

Definíció a(z) **datum.h** fájl **17**. sorában.

◆ ho

int Datum::ho private

Definíció a(z) **datum.h** fájl **18**. sorában.

◆ nap

int Datum::nap — private

Definíció a(z) **datum.h** fájl **19**. sorában.

4.3.4. String osztály

Ez az osztály a dátum osztály kivételével minden osztályban használva van. Csak az általam használt stringkezelő függvények lettek megvalósítva benne, ezért nem egy teljes értékű stringkezelő osztály.

Publikus tagfüggvények

	String () Paraméter nélküli konstruktor: Részletek...
size_t	**size** () const
const char *	**c_str** () const
	String (const char *) Konstruktor: egy nullával lezárt char sorozatból. Részletek...
	String (const **String** &)
	~String () Destruktor. Részletek...
String &	**operator=** (const **String** &)
bool	**operator==** (const **String** &) const
bool	**operator!=** (const **String** &) const

Privát attribútumok

char *	**pData**
size_t	**len**

Háromféle konstruktorból áll. A paraméter nélkül hívható konstruktor üres karaktersorozatot hoz létre és a hosszát jelző egész típusú adattagot 0 értékre állítja. A paraméteres konstruktora egy bemeneti paraméterként kapott karaktersorozat pointerét állítja be a Pdata adattagnak. Megszámolja a karaktersorozat hosszát és ezzel az értékkel inicializálja a len nevű adattagot. A harmadik konstruktor a másoló konstruktor.

Konstruktorok és destruktorok dokumentációja

◆ String() [1/3]

String::String ( ) — inline

Paraméter nélküli konstruktor:

Definíció a(z) **string.h** fájl **22**. sorában.

◆ String() [2/3]

String::String (const char * p)

Konstruktor: egy nullával lezárt char sorozatból.

Definíció a(z) **string.cpp** fájl **21**. sorában.

◆ String() [3/3]

String::String (const **String** & s)

MÁSOLÓ konstruktor

Paraméterek

s1 - **String**, amiből létrehozzuk az új String-et

Definíció a(z) **string.cpp** fájl **27**. sorában.

◆ ~String()

String::~String ( ) inline

Destruktor.

Definíció a(z) **string.h** fájl **43**. sorában.

Tagfüggvények dokumentációja

◆ c_str()

const char * String::c_str ( ) const inline

C-sztringet ad vissza

Visszatérési érték

pointer a tárolt, vagy azzal azonos tartalmú nullával lezárt sztring-re.

Definíció a(z) **string.h** fájl **33**. sorában.

A c_str függvény egy C típusú stringre mutató pointerrel tér vissza.

A komparátorok két String egyezőségét, illetve nem egyezőségét vizsgálják. Először megvizsgálja a hosszukat, majd karakterről karakterre megnézi az egyezőséget.

◆ operator!=()

bool String::operator!= (const **String** & s) const

Két **String** nem egyezőségét vizsgálja

Paraméterek

s - jobb oldali operandus

Visszatérési érték

true, ha nem egyezik a két **String**

Definíció a(z) **string.cpp** fájl **50**. sorában.

◆ operator=()

String & String::operator= (const **String** & s)

Értékadó operátor

Paraméterek

rhs_s - jobboldali **String**

Visszatérési érték

baoldali (módosított) string (referenciája)

Definíció a(z) **string.cpp** fájl **32**. sorában.

◆ operator==()

bool String::operator== (const **String** & s) const

Két **String** egyezőségét vizsgálja

Paraméterek

s - jobb oldali operandus

Visszatérési érték

true, ha egyezik a két **String**

Definíció a(z) **string.cpp** fájl **41**. sorában.

◆ size()

size_t String::size ( ) const inline

Sztring hosszát adja vissza.

Visszatérési érték

sztring tényleges hossza (lezáró nulla nélkül).

Definíció a(z) **string.h** fájl **29**. sorában.

Adattagok dokumentációja

◆ len

size_t String::len — private

Definíció a(z) **string.h** fájl **19**. sorában.

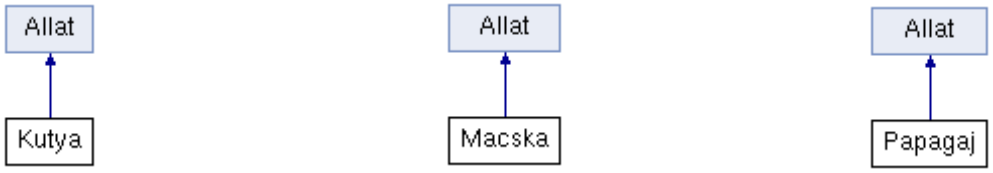
◆ pData

char* String::pData — private

Definíció a(z) **string.h** fájl **18**. sorában.

4.3.5. Kutya, macska és papagáj osztályok

Ezek az állat osztályból származtatott osztályok belső felépítésükben teljesen megegyeznek így csak a kutya osztályt mutatom be részletesen. Származási diagramjaik:



Privát attribútumok

Tulaj	**gazdi**
String	**chip**
String	**bunda**
bool	**nem**

További örökölt tagok

Védett attribútumok a(z) Allat osztályból származnak

String	**faj**
String	**nev**
Datum	**szul**

Az osztálynak kétféle konstruktora van.

A paraméter nélkül hívható konstruktor minden adattagot üres tartalomra állít be, a dátumot és a nemet kivéve. A dátum az aktuális dátum lesz, a nem adattag pedig hamis értékű.

A paraméteres konstruktor hívásakor kötelező minden adattagot megadni. A dátum típusú adattagra a dátum osztály ellenőrzése, a tulaj típusú adattagra a tulaj osztály ellenőrzése fut le. Ezek alapján dobhat hibát a konstruktor. A string típusú adattag megadásakor az üres tartalmú string megadásakor hibát dob a konstruktor.

Publikus tagfüggvények

	Kutya () Default konstruktor. Részletek...
	Kutya (const char ***faj**, const char ***nev**, const **Datum** &**szul**, const **Tulaj** &**gazdi**, const char ***chip**, const char ***bunda**, bool **nem**)
bool	**operator==** (const **Kutya** &) const
bool	**operator!=** (const **Kutya** &) const
Tulaj	**getTulaj** ()
void	**setTulaj** (**Tulaj** &t)
String	**getChip** ()
void	**setChip** (const char *p)
String	**getBunda** ()
void	**setBunda** (const char *p)
bool	**getNem** ()
void	**setNem** (bool b)
	~Kutya () Destruktor. Részletek...

Publikus tagfüggvények a(z) Allat osztályból származnak

	Allat () Default konstruktor. Részletek...
	Allat (const char ***faj**, const char ***nev**, const **Datum** &**szul**)
String	**getFaj** () const
void	**setFaj** (const char *p)
String	**getNev** () const
void	**setNev** (const char *p)
Datum	**getSzul** () const
void	**setSzul** (**Datum** &d)
virtual	**~Allat** () Virtuális destruktor. Részletek...

Konstruktorok és destruktorok dokumentációja

◆ Kutya() [1/2]

Kutya::Kutya ( ) inline

Default konstruktor.

Definíció a(z) **kutya.h** fájl **25**. sorában.

◆ Kutya() [2/2]

```
Kutya::Kutya ( const char *      faj,
               const char *      nev,
               const Datum &     szul,
               const Tulaj &     gazdi,
               const char *      chip,
               const char *      bunda,
               bool              nem
             )
```

Konstruktor beállítja az attribútumokat

Paraméterek

- **faj** - faj megnevezése
- **nev** - név megadása
- **szul** - születési év megadása
- **gazdi** - tulaj megadása
- **chip** - chipszám megadása
- **bunda** - bunda jellege, színe megadása
- **nem** - nem megadása

Definíció a(z) **kutya.cpp** fájl **12**. sorában.

◆ ~Kutya()

```
Kutya::~Kutya ( )
```
inline

Destruktor.

Definíció a(z) **kutya.h** fájl **90**. sorában.

Tagfüggvények dokumentációja

◆ getBunda()

```
String Kutya::getBunda ( )
```
inline

Bunda lekérdezése

Visszatérési érték

- bunda

Definíció a(z) **kutya.h** fájl **70**. sorában.

◆ getChip()

String Kutya::getChip ( ) inline

Chipszám lekérdezése

Visszatérési érték

- chip

Definíció a(z) **kutya.h** fájl **58**. sorában.

◆ getNem()

bool Kutya::getNem ( ) inline

Nem lekérdezése True - nőstény, False - hím

Visszatérési érték

- nem

Definíció a(z) **kutya.h** fájl **83**. sorában.

◆ getTulaj()

Tulaj Kutya::getTulaj ( ) inline

Tulaj lekérdezése

Visszatérési érték

- gazdi

Definíció a(z) **kutya.h** fájl **50**. sorában.

A getBunda getChip nevű függvények konstans string típusú megfelelő adattagok értékével térnek vissza. A getTulaj nevű függvény egy Tulaj típussal tér vissza.

◆ operator!=()

bool Kutya::operator!= (const **Kutya** & k) const

Két kutya nem egyezőségét vizsgálja

Paraméterek

k - jobb oldali operandus

Visszatérési érték

true, ha nem egyezik a két kutya

Definíció a(z) **kutya.cpp** fájl **26**. sorában.

◆ operator==()

bool Kutya::operator== (const **Kutya** & k) const

Két kutya egyezőségét vizsgálja

Paraméterek

k - jobb oldali operandus

Visszatérési érték

true, ha egyezik a két kutya

Definíció a(z) **kutya.cpp** fájl **21**. sorában.

Az összehasonlító operátorok két kutya típusú példány egyezőségét, illetve nem egyezőségét vizsgálja. Tulaj és dátum típus esetén azok összehasonlító operátorai hívódnak meg. Visszatérési értékük logikai típus.

◆ setBunda()

void Kutya::setBunda (const char * p) inline

Bundázat beállítása

Paraméterek

p - karaktersorozat

Definíció a(z) **kutya.h** fájl **74**. sorában.

◆ setChip()

void Kutya::setChip (const char * p) inline

Chipszám beállítása

Paraméterek

p - karaktersorozat

Definíció a(z) **kutya.h** fájl **62**. sorában.

◆ setNem()

void Kutya::setNem (bool b) inline

Nem beállítása

Paraméterek

b - true/false

Definíció a(z) **kutya.h** fájl **87**. sorában.

◆ setTulaj()

void Kutya::setTulaj (**Tulaj** & t) inline

Tulaj beállítása

Paraméterek

t - **Tulaj** referencia

Definíció a(z) **kutya.h** fájl **54**. sorában.

Setterek beállítják a megfelelő adattagok értékét. A setBunda és setChip függvények bemeneti paramétere egy karaktersorozat, ha ez nem üres tartalmú, akkor beállításra kerülnek, ha üresek, akkor hibaát dobnak a függvények. A setNem függvény egy logikai típust vár bemeneti paraméterként. A setTulaj egy Tulaj típusú adatot vár bemeneti paraméternek. A bemeneti paraméter ellenőrzése nem szükséges, mivel a paraméter létrehozásakor már megtörténik az adatok helyes megadása.

Adattagok dokumentációja

◆ bunda

String Kutya::bunda private

Definíció a(z) **kutya.h** fájl **21**. sorában.

◆ chip

String Kutya::chip private

Definíció a(z) **kutya.h** fájl **20**. sorában.

◆ gazdi

Tulaj Kutya::gazdi private

Definíció a(z) **kutya.h** fájl **19**. sorában.

◆ nem

bool Kutya::nem private

Definíció a(z) **kutya.h** fájl **22**. sorában.

4.4. Tesztprogram bemutatása

A tesztelés a NagyHazi.cpp nevű fájlban történik a gtest_lite.h nevű forrásfájl felhasználásával.

Függvények

void	**test_1** () TESZT 1. - létrehozás + beszúrás + operátorok + setterek tesztelése. Részletek...
void	**test_2** () TESZT 2. - kiírás tesztelése. Részletek...
void	**test_3** () TESZT 3. - fájlba való kiírás tesztelése. Részletek...
void	**test_4** () TESZT 4. - törlés tesztelése. Részletek...
void	**test_5** () TESZT 5. - módosítás tesztelése. Részletek...
void	**test_6** () TESZT 6. - fájlból való beolvasás tesztelése. Részletek...
int	**main** ()

4.4.1. Teszteseteket megvalósító függvények

◆ main()

int main ( )

Definíció a(z) **NagyHazi.cpp** fájl **552**. sorában.

◆ test_1()

void test_1 ( )

TESZT 1. - létrehozás + beszúrás + operátorok + setterek tesztelése.

Definíció a(z) **NagyHazi.cpp** fájl **16**. sorában.

◆ test_2()

void test_2 ( )

TESZT 2. - kiírás tesztelése.

Definíció a(z) **NagyHazi.cpp** fájl **356**. sorában.

◆ test_3()

void test_3 ( )

TESZT 3. - fájlba való kiírás tesztelése.

Definíció a(z) **NagyHazi.cpp** fájl **391**. sorában.

◆ test_4()

void test_4 ( )

TESZT 4. - törlés tesztelése.

Definíció a(z) **NagyHazi.cpp** fájl **453**. sorában.

◆ test_5()

void test_5 ( )

TESZT 5. - módosítás tesztelése.

Definíció a(z) **NagyHazi.cpp** fájl **491**. sorában.

◆ test_6()

void test_6 ( )

TESZT 6. - fájlból való beolvasás tesztelése.

Definíció a(z) **NagyHazi.cpp** fájl **525**. sorában.

5. Tesztelés

5.2. A funkcionális teszt

A program minden egyes funkcióját leteszteltem jó, illetve rossz bemenetekkel is, hogy bemutassam a megfelelő működését. A gtest_lite.h nevű fájlból a következő függvényeket használtam fel a megfelelő kimenetek ellenőrzésére: EXPECT_NO_THROW, EXPECT_ANY_THROW, EXPECT_STREQ.

5.3. Memóriakezelés tesztje

A memóriakezelés ellenőrzését a laborgyakorlatokon is használt memtrace.h és memtrace.cpp fájlok felhasználásával végeztem. Minden önálló fordítási egységben include-oltam a memtrace.h fájlt. A futtatások során nem tapasztaltam memóriakezelési hibát.

5.4. Lefedettségi teszt

A funkcionális tesztek a program minden ágát lefedték. A Jporta 2 programsort jelölt meg, ami nem futott a programban. Az egyik egy olyan hiba, amit a program megvalósításából adódóan nem lehet előidézni. A másik egy olyan részlet, ami nem futott le a tesztek futása során.

```
106.        /// Iterátor konstruktor
107.        /// Elejére állítja az iterátort
108.        /// @param l - lista refernciája
109.        iterator(const Lista& l) : akt(l.elso) {
110.            if (akt->kov == NULL)
111.                akt = NULL;                 // strázsa miatti trükk
112.        }
```

```
147.        /// Indirekció
148.        /// @return osztály pointer
149.        T* operator->() {
150.            if (akt != NULL) return(&akt->adat);    // visszatérés az adattal
151.            else throw std::out_of_range("Hiba");
152.        }
```