Garbage Collector - Dremák Gergely

EREDETI SPECIFIKÁCIÓ

Leírás

3 fajta "okos" pointer osztály a <memory> analógiájára:

- **Unique pointer**: Egyetlen példány van belőle, amint kikerül a scope-ból feltakarít maga után.
- **Shared pointer**: Egy counter van a háttérben ami számon tartja a referenciák számát a memória területre. Ha ez 0 a memória felszabadul.
- **Weak pointer**: Effektíve egy átlagos pointer, de meg lehet kérdezni tőle, hogy a memória amire mutat tartalmaz-e még értelemes adatot (törölve lett-e), illetve lehet promotálni a másik 2 pointer típussá.

Deklarálás példa (a név és szintaxis változtatás jogát fenntartom):

```
class T {
      T(…args);
      void doSomething();
};
int main(void) {
  SharedPointer<T> ptr = SharedPointer::Init<T>(…args);
  // vagy
  SharedPointer<T> ptr(new T(…args));

  ptr->doSomething();
  // <- Destruktor hívás
}
```

Tulajdonságok

- ~~STL kollekciók~~

Teszt

A *memtrace* a nagyrészét intézi, nyilván a shared pointer a trükkösebb, mert lehet, hogy több thread is használja illetve függvényekben megfelelően inicializálódik = növekszik-e a counter.

Egyéb kérdés esetén a <memory> könyvárral történő konzultálást javasolom mert sok lesz a párhuzam.

EREDETI SPECIFIKÁCIÓ VÉGE

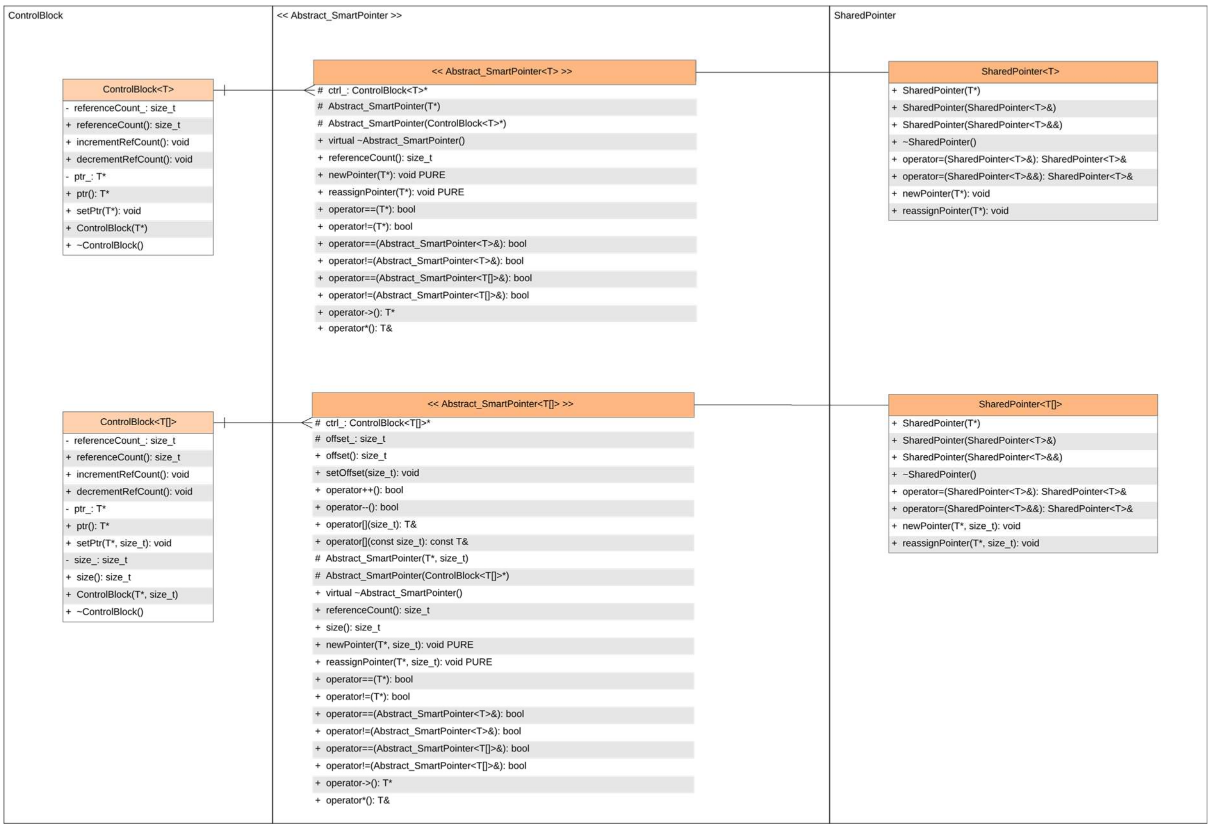
Dremák Gergely – KSHSLY
Garbage Collector

Class Hierarchy

CHF2::Abstract_SmartPointer< T >	Skaláris okospointer alaposztály
CHF2::SharedPointer< T >	Autómatikusan törlődő pointer
CHF2::Abstract_SmartPointer< T[]>	Vektorális okospointer alaposztály
CHF2::SharedPointer< T[]>	Autómatikusan törlődő tömb pointer
CHF2::ControlBlock< T >	Skaláris Pointer tárolására
CHF2::ControlBlock< T[]>	Vektorális Pointer tárolására

File List

▼ NHF2
garbage_collector_test.cpp
hf2.hpp
hf2_control_block.hpp
hf2_shared_ptr.cpp
hf2_shared_ptr.hpp
hf2_smart_ptr.hpp


ControlBlock
<< Abstract_SmartPointer >>
SharedPointer
ControlBlock<T>
- referenceCount_: size_t
+ referenceCount(): size_t
+ incrementRefCount(): void
+ decrementRefCount(): void
- ptr_: T*
+ ptr(): T*
+ setPtr(T*): void
+ ControlBlock(T*)
+ ~ControlBlock()
<< Abstract_SmartPointer<T> >>
# ctrl_: ControlBlock<T>*
# Abstract_SmartPointer(T*)
# Abstract_SmartPointer(ControlBlock<T>*)
+ virtual ~Abstract_SmartPointer()
+ referenceCount(): size_t
+ newPointer(T*): void PURE
+ reassignPointer(T*): void PURE
+ operator==(T*): bool
+ operator!=(T*): bool
+ operator==(Abstract_SmartPointer<T>&): bool
+ operator!=(Abstract_SmartPointer<T>&): bool
+ operator==(Abstract_SmartPointer<T[]>&): bool
+ operator!=(Abstract_SmartPointer<T[]>&): bool
+ operator->(): T*
+ operator*(): T&
SharedPointer<T>
+ SharedPointer(T*)
+ SharedPointer(SharedPointer<T>&)
+ SharedPointer(SharedPointer<T>&&)
+ ~SharedPointer()
+ operator=(SharedPointer<T>&): SharedPointer<T>&
+ operator=(SharedPointer<T>&&): SharedPointer<T>&
+ newPointer(T*): void
+ reassignPointer(T*): void
ControlBlock<T[]>
- referenceCount_: size_t
+ referenceCount(): size_t
+ incrementRefCount(): void
+ decrementRefCount(): void
- ptr_: T*
+ ptr(): T*
+ setPtr(T*, size_t): void
- size_: size_t
+ size(): size_t
+ ControlBlock(T*, size_t)
+ ~ControlBlock()
<< Abstract_SmartPointer<T[]> >>
# ctrl_: ControlBlock<T[]>*
# offset_: size_t
+ offset(): size_t
+ setOffset(size_t): void
+ operator++(): bool
+ operator--(): bool
+ operator[](size_t): T&
+ operator[](const size_t): const T&
# Abstract_SmartPointer(T*, size_t)
# Abstract_SmartPointer(ControlBlock<T[]>*)
+ virtual ~Abstract_SmartPointer()
+ referenceCount(): size_t
+ size(): size_t
+ newPointer(T*, size_t): void PURE
+ reassignPointer(T*, size_t): void PURE
+ operator==(T*): bool
+ operator!=(T*): bool
+ operator==(Abstract_SmartPointer<T>&): bool
+ operator!=(Abstract_SmartPointer<T>&): bool
+ operator==(Abstract_SmartPointer<T[]>&): bool
+ operator!=(Abstract_SmartPointer<T[]>&): bool
+ operator->(): T*
+ operator*(): T&
SharedPointer<T[]>
+ SharedPointer(T*)
+ SharedPointer(SharedPointer<T>&)
+ SharedPointer(SharedPointer<T>&&)
+ ~SharedPointer()
+ operator=(SharedPointer<T>&): SharedPointer<T>&
+ operator=(SharedPointer<T>&&): SharedPointer<T>&
+ newPointer(T*, size_t): void
+ reassignPointer(T*, size_t): void

HF2::ControlBlock< T > Class Template Reference

Skaláris Pointer tárolására.

#include <**hf2_control_block.hpp**>

Public Member Functions

	ControlBlock (T *t) explicit konstruktor
	~ControlBlock () noexcept destruktor
T *	**ptr** () const ptr_ getter
void	**setPtr** (T ***ptr**) ptr_ setter
std::size_t	**referenceCount** () const referenceCount_ getter
void	**incrementRefCount** () const inkrementálja a referenceCount_ értékét
void	**decrementRefCount** () const dekrementálja a referenceCount_ értékét

Detailed Description

template<class T> class HF2::ControlBlock< T >

Skaláris Pointer tárolására.

Definition at line **15** of file **hf2_control_block.hpp**.

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

- **hf2_control_block.hpp**

HF2::ControlBlock< T[]> Class Template Reference

Vektorális Pointer tárolására.

#include <**hf2_control_block.hpp**>

Public Member Functions

	ControlBlock (T *t, std::size_t **size**) explicit konstruktor
	~ControlBlock () noexcept Destruktor.
T *	**ptr** () const ptr_ getter
void	**setPtr** (T ***ptr**, std::size_t **size**) ptr_ setter
std::size_t	**referenceCount** () const referenceCount_ getter
void	**incrementRefCount** () const inkrementálja a referenceCount_ értékét
void	**decrementRefCount** () const dekrementálja a referenceCount_ értékét
std::size_t	**size** () const size_ getter

Detailed Description

template<class T>
class HF2::ControlBlock< T[]>
Vektorális Pointer tárolására.
A setPtr<T[]>(...) potenciálisan a blokkot használó objektumok offsetjének a túlindexelését jelentheti, exception-t meg inkább nem dobatok vele mert általános használat közben könnyen elkerülhető egy ilyen probléma, feliratkozós; callbackes; event emitteres; frissítés meg már sok lenne ide. Szóval bízok a felhasználóban. Mellesleg javallott a ++/– operátorokat csak loopokon belül használni és a végén 0-ba állítani
Definition at line **74** of file **hf2_control_block.hpp**.

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

- F:/Programming/Uni/Prog2/NHF/NHF2/**hf2_control_block.hpp**

Dremák Gergely – KSHSLY
Garbage Collector

HF2::Abstract_SmartPointer< T > Class Template Reference abstract

Skaláris okospointer alaposztály.

#include <**hf2_smart_ptr.hpp**>

▸ Inheritance diagram for HF2::Abstract_SmartPointer< T >:

Public Member Functions

virtual	**~Abstract_SmartPointer** () virtuális destruktor, hogy biztosan meghívódjon a leszármazotté
std::size_t	**referenceCount** () const ctrl_->**referenceCount()** facade
virtual void	**newPointer** (T *t)=0 új **ControlBlock** inicializálása a példánynak.
virtual void	**reassignPointer** (T *t)=0 új pointer beállítása az összes pointernek ami ezt a **ControlBlock** -ot használja.
T *	**operator->** () arrow
T &	**operator*** () dereferálás
bool	**operator==** (const T *t) const t == ctrl_->t
bool	**operator!=** (const T *t) const t != ctrl_->t
bool	**operator==** (const **Abstract_SmartPointer**< T > &sp) const sp.ctrl_->t != ctrl_->t
bool	**operator!=** (const **Abstract_SmartPointer**< T > &sp) const sp.ctrl_->t != ctrl_->t
bool	**operator==** (const **Abstract_SmartPointer**< T[]> &sp) const sp.ctrl_->t + offset == ctrl_->t
bool	**operator!=** (const **Abstract_SmartPointer**< T[]> &sp) const sp.ctrl_->t + offset != ctrl_->t

Protected Member Functions

Abstract_SmartPointer (T *t)
"új" pointer inicializálás

Abstract_SmartPointer (**ControlBlock**< T > *ctrl)
létező pointerrel osztozkodás

Protected Attributes

ControlBlock< T > *	**ctrl_** = nullptr

Pointer kontrollblokkra.

Detailed Description

template<class T>

class HF2::Abstract_SmartPointer< T >

Skaláris okospointer alaposztály.

Definition at line **13** of file **hf2_smart_ptr.hpp**.

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

- F:/Programming/Uni/Prog2/NHF/NHF2/**hf2_smart_ptr.hpp**

Dremák Gergely – KSHSLY
Garbage Collector

HF2::Abstract_SmartPointer< T[]> Class Template Reference abstract

Vektorális okospointer alaposztály.

#include <**hf2_smart_ptr.hpp**>

Inheritance diagram for HF2::Abstract_SmartPointer< T[]>:

Public Member Functions

virtual	**~Abstract_SmartPointer** () virtuális destruktor, hogy biztosan meghívódjon a leszármazotté
std::size_t	**referenceCount** () const ctrl_->**referenceCount()** facade
std::size_t	**size** () const ctrl_->**size()** facade
std::size_t	**offset** () const offset_ getter
void	**setOffset** (std::size_t **offset**) const offset_ setter
virtual void	**newPointer** (T *t, std::size_t **size**)=0 új **ControlBlock<T[]>** inicializálása a példánynak.
virtual void	**reassignPointer** (T *t, std::size_t **size**)=0 új pointer beállítása az összes pointernek ami ezt a **ControlBlock<T[]>** -ot használja.
T &	**operator[]** (const std::size_t idx) indexelés 0. elemtől | const *char exception-t dob túlindexelésnél
const T &	**operator[]** (const std::size_t idx) const Pont arra jó mint a másik csak ez még const is.
bool	**operator++** () const ++offset_ | nincs post-increment | ha eléri a ctrl_->**size()** -t akkor 0-ra áll vissza és false-t dob
bool	**operator--** () const –offset_ | nincs post-decrement | ha elérné a -1 -t akkor **size()** - 1-re áll vissza és false-t dob
T *	**operator->** () const jelenleg offsettel kiválaszott elem tagjának hozzáférése
T &	**operator*** () const jelenleg offsettel kiválaszott elem dereferálása
bool	**operator==** (const T *t) const t != ctrl_->t + offset
bool	**operator!=** (const T *t) const t != ctrl_->t + offset

bool	**operator==** (const **Abstract_SmartPointer**< T > &sp) const sp.ctrl_->t == ctrl_->t + offset
bool	**operator!=** (const **Abstract_SmartPointer**< T > &sp) const sp.ctrl_->t != ctrl_->t + offset
bool	**operator==** (const **Abstract_SmartPointer**< T[]> &sp) const sp.ctrl_->t + sp.offset == ctrl_->t + offset
bool	**operator!=** (const **Abstract_SmartPointer**< T[]> &sp) const sp.ctrl_->t + sp.offset != ctrl_->t + offset

Protected Member Functions

Abstract_SmartPointer (T *t, std::size_t **size**)
"új" pointer inicializálás

Abstract_SmartPointer (**ControlBlock**< T[]> *ctrl)
létező pointerrel osztozkodás

Protected Attributes

ControlBlock< T[]> *	**ctrl_** = nullptr Pointer kontrollblokkra.
std::size_t	**offset_** = 0U Bármikor módosítható offset.

Detailed Description

template<class T>
class HF2::Abstract_SmartPointer< T[]>
Vektorális okospointer alaposztály.
javallott a ++/– operátorokat csak loopokon belül használni és a végén 0-ba állítani
Definition at line **66** of file **hf2_smart_ptr.hpp**.

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

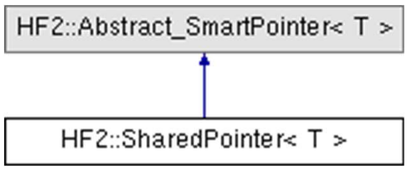
- F:/Programming/Uni/Prog2/NHF/NHF2/**hf2_smart_ptr.hpp**

HF2::SharedPointer< T > Class Template Reference

Autómatikusan törlődő pointer.

#include <**hf2_shared_ptr.hpp**>

Inheritance diagram for HF2::SharedPointer< T >:



Public Member Functions

	SharedPointer (T *t) "új" pointer inicializálás
	SharedPointer (**SharedPointer**< T > &&sp) noexcept mozgató konstruktor
	SharedPointer (const **SharedPointer**< T > &sp) másoló konstruktor
	~SharedPointer () destruktor
void	**newPointer** (T *t) override új **ControlBlock** inicializálása a példánynak.
void	**reassignPointer** (T *t) override új pointer beállítása az összes pointernek ami ezt a **ControlBlock** -ot használja.
SharedPointer< T > &	**operator=** (**SharedPointer**< T > &&sp) mozgató értékadás
SharedPointer< T > &	**operator=** (const **SharedPointer**< T > &sp) másoló értékadás

Public Member Functions inherited from **HF2::Abstract_SmartPointer< T >**

virtual	**~Abstract_SmartPointer** () virtuális destruktor, hogy biztosan meghívódjon a leszármazotté
std::size_t	**referenceCount** () const ctrl_->**referenceCount()** facade
T *	**operator->** () arrow
T &	**operator*** () dereferálás
bool	**operator==** (const T *t) const t == ctrl_->t
bool	**operator!=** (const T *t) const

	t != ctrl_->t
bool	**operator==** (const **Abstract_SmartPointer**< T > &sp) const sp.ctrl_->t != ctrl_->t
bool	**operator!=** (const **Abstract_SmartPointer**< T > &sp) const sp.ctrl_->t != ctrl_->t
bool	**operator==** (const **Abstract_SmartPointer**< T[]> &sp) const sp.ctrl_->t + offset == ctrl_->t
bool	**operator!=** (const **Abstract_SmartPointer**< T[]> &sp) const sp.ctrl_->t + offset != ctrl_->t

Additional Inherited Members

Protected Member Functions inherited from **HF2::Abstract_SmartPointer< T >**

	Abstract_SmartPointer (T *t) "új" pointer inicializálás
	Abstract_SmartPointer (**ControlBlock**< T > *ctrl) létező pointerrel osztozkodás

Protected Attributes inherited from **HF2::Abstract_SmartPointer< T >**

ControlBlock< T > *	**ctrl_** = nullptr Pointer kontrollblokkra.

Detailed Description

template<class T>
class HF2::SharedPointer< T >
Autómatikusan törlődő pointer.
Lehetőleg *ne* dinamikusan allokáljuk. Mozgató konstruktorok és operátorok az std::move()-val használhatók
Definition at line **16** of file **hf2_shared_ptr.hpp**.

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

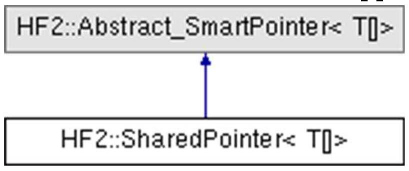
- F:/Programming/Uni/Prog2/NHF/NHF2/**hf2_shared_ptr.hpp**
- F:/Programming/Uni/Prog2/NHF/NHF2/**hf2_shared_ptr.cpp**

HF2::SharedPointer< T[]> Class Template Reference

Autómatikusan törlődő tömb pointer.

#include <**hf2_shared_ptr.hpp**>

Inheritance diagram for HF2::SharedPointer< T[]>:



Public Member Functions

	SharedPointer (T *t, std::size_t **size**) "új" pointer inicializálás
	SharedPointer (**SharedPointer**< T[]> &&sp) noexcept mozgató konstruktor
	SharedPointer (const **SharedPointer**< T[]> &sp) másoló konstruktor
	~SharedPointer () destruktor
void	**newPointer** (T *t, std::size_t **size**) override új **ControlBlock<T[]>** inicializálása a példánynak.
void	**reassignPointer** (T *t, std::size_t **size**) override új pointer beállítása az összes pointernek ami ezt a **ControlBlock<T[]>** -ot használja.
SharedPointer< T[]> &	**operator=** (**SharedPointer**< T[]> &&sp) noexcept mozgató értékadás
SharedPointer< T[]> &	**operator=** (const **SharedPointer**< T[]> &sp) másoló értékadás

Public Member Functions inherited from **HF2::Abstract_SmartPointer< T[]>**

virtual	**~Abstract_SmartPointer** () virtuális destruktor, hogy biztosan meghívódjon a leszármazotté
std::size_t	**referenceCount** () const ctrl_->**referenceCount()** facade

std::size_t	**size** () const ctrl_->**size()** facade
std::size_t	**offset** () const offset_ getter
void	**setOffset** (std::size_t **offset**) const offset_ setter
T &	**operator[]** (const std::size_t idx) indexelés 0. elemtől | const *char exception-t dob túlindexelésnél
const T &	**operator[]** (const std::size_t idx) const Pont arra jó mint a másik csak ez még const is.
bool	**operator++** () const ++offset_ | nincs post-increment | ha eléri a ctrl_->**size()** -t akkor 0-ra áll vissza és false-t dob
bool	**operator--** () const –offset_ | nincs post-decrement | ha elérné a -1 -t akkor **size()** - 1-re áll vissza és false-t dob
T *	**operator->** () const jelenleg offsettel kiválaszott elem tagjának hozzáférése
T &	**operator*** () const jelenleg offsettel kiválaszott elem dereferálása
bool	**operator==** (const T *t) const t != ctrl_->t + offset
bool	**operator!=** (const T *t) const t != ctrl_->t + offset
bool	**operator==** (const **Abstract_SmartPointer**< T > &sp) const sp.ctrl_->t == ctrl_->t + offset
bool	**operator!=** (const **Abstract_SmartPointer**< T > &sp) const sp.ctrl_->t != ctrl_->t + offset

bool	**operator==** (const **Abstract_SmartPointer**< T[]> &sp) const sp.ctrl_->t + sp.offset == ctrl_->t + offset
bool	**operator!=** (const **Abstract_SmartPointer**< T[]> &sp) const sp.ctrl_->t + sp.offset != ctrl_->t + offset

Additional Inherited Members

Protected Member Functions inherited from **HF2::Abstract_SmartPointer< T[]>**

	Abstract_SmartPointer (T *t, std::size_t **size**) "új" pointer inicializálás
	Abstract_SmartPointer (**ControlBlock**< T[]> *ctrl) létező pointerrel osztozkodás

Protected Attributes inherited from **HF2::Abstract_SmartPointer< T[]>**

ControlBlock< T[]> *	**ctrl_** = nullptr Pointer kontrollblokkra.
std::size_t	**offset_** = 0U Bármikor módosítható offset.

Detailed Description

template<class T>
class HF2::SharedPointer< T[]>

Autómatikusan törlődő tömb pointer.

Lehetőleg *ne* dinamikusan allokáljuk. Mozgató konstruktorok és operátorok az std::move()-val használhatók

Definition at line **70** of file **hf2_shared_ptr.hpp**.

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

- F:/Programming/Uni/Prog2/NHF/NHF2/**hf2_shared_ptr.hpp**
- F:/Programming/Uni/Prog2/NHF/NHF2/**hf2_shared_ptr.cpp**