

Programozás alapjai 2

NHF Specifikáció(Bővített)

Fótyék Róbert

DNHR9M

Leírás: A program egy részecskék közötti kölcsönhatásokat modellező egyszerű, kétdimenziós szimuláció. A kezdeti állapot beolvasás után képes tovább léptetni a szimulációt tetszőleges időtartamig. A nagyházi követelményeit figyelembe véve *nem* lesz grafikus felülete, a kimenete a részecskék pillanatnyi tulajdonságai, melyet a program egy fájlba ír.

I/O: A program szimuláció közben minden időegységben egy, az alábbi formátumú fájlba írja a részecskék adatait (ez lehetővé teszi egy grafikus "visszajátszó" program elkészítését). A bemenet egy ugyanilyen formátumú fájl, és a program a fájlban található utolsó időegységről folytatja a szimulációt, tovább írva a fájlt.

Feature-ök: egy részecske deklarálásakor mindig megadandó a tömege, a töltése, illetve beállítható hogy részt vesz-e a gravitációs kölcsönhatásban. Ezen konstansokon kívül beállítható kezdetleges pozíciója és koordinátái. Ilyen részecskékből tetszőleges számú megadható. Ez a kezdeti állapot megadható fájlban, vagy a program CLI-jén keresztül.

Fájl formátum:

{

Tömeg,

PozícióX PozícióY,

SebességX SebességY,

Töltés,

GravitációsBoolean,

Tömeg,

PozícióX PozícióY,

SebességX SebességY,

Töltés,

GravitációsBoolean,

...

...

...

}

{

...

}

.

.

.

Így egy bracket {} egy időegység, és azon belül a részecskék a szabvány alapján következnek egymás után. A sablonban az új sor csak a szépség miatt van, egyébként minden whitespace ugyanaz.

Program futtatása:

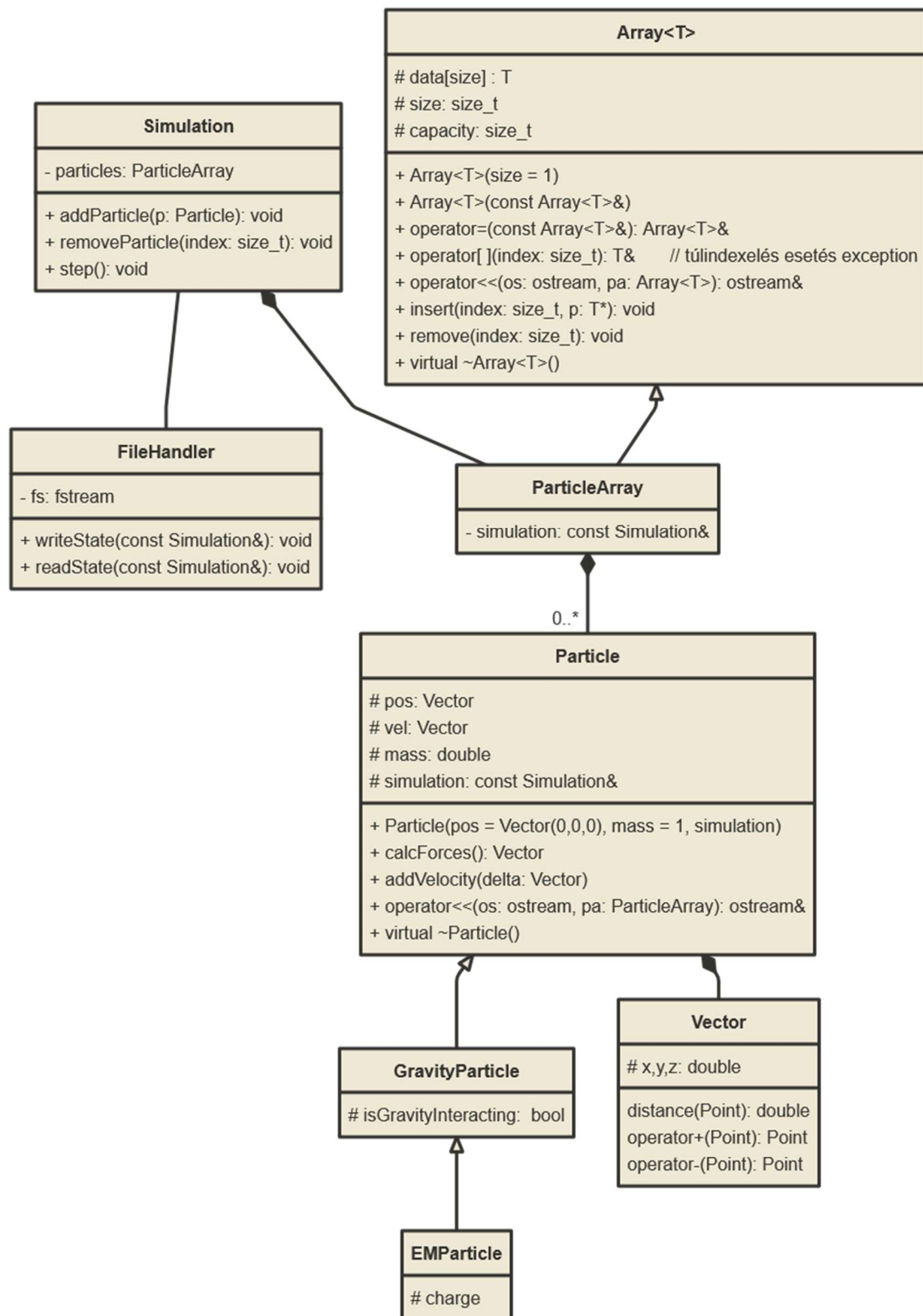
A program argumentumként megkapja az input fájlt, majd onnan menürendszerből kezelhető.

Menüpontok:

- ListParticles
- AddParticle
- RemoveParticle
- Simulate *időtartam*
- *Exit*

A szimulált időtartam után visszalép a program a menübe.

Osztálydiagram



Megjegyzések:

- A ParticleArray illetve Particle osztályoknak komponense a Simulation osztály, hogy ne lehessen részecske hozzá tartozó szimuláció nélkül.
- Accesorokat, triviális konstruktorokat, destruktorkat nem jelöltem.

Fő algoritmus: szimulációs lépés kiszámítása

```
For each Particle i in ParticleArray:  
    Vector force(0,0,0)  
    For each Particle j != i in ParticleArray:  
        Vector force += calculateEMforce(i, j)  
    i.addVelocity(force/i.mass())
```

Particular

Generated by Doxygen 1.13.2

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 Array< T > Class Template Reference	5
3.1.1 Detailed Description	7
3.1.2 Constructor & Destructor Documentation	7
3.1.2.1 Array() [1/2]	7
3.1.2.2 Array() [2/2]	8
3.1.3 Member Function Documentation	8
3.1.3.1 getCapacity()	8
3.1.3.2 getSize()	8
3.1.3.3 insert() [1/2]	8
3.1.3.4 insert() [2/2]	8
3.1.3.5 operator=()	9
3.1.3.6 operator==()	9
3.1.3.7 operator[]() [1/2]	9
3.1.3.8 operator[]() [2/2]	10
3.1.3.9 read()	10
3.1.3.10 remove()	10
3.1.3.11 write()	11
3.2 Particle Class Reference	11
3.2.1 Detailed Description	13
3.2.2 Constructor & Destructor Documentation	13
3.2.2.1 Particle()	13
3.2.3 Member Function Documentation	14
3.2.3.1 applyForce()	14
3.2.3.2 forceWith()	14
3.2.3.3 getCharge()	14
3.2.3.4 getGrav()	14
3.2.3.5 getMass()	15
3.2.3.6 getPos()	15
3.2.3.7 getVel()	15
3.2.3.8 move()	15
3.2.3.9 operator==()	15
3.2.3.10 read()	16
3.2.3.11 write()	16
3.3 Simulation Class Reference	16
3.3.1 Detailed Description	18
3.3.2 Member Function Documentation	18

3.3.2.1 addParticle()	18
3.3.2.2 getParticles()	18
3.3.2.3 listParticles()	18
3.3.2.4 operator==()	18
3.3.2.5 read()	19
3.3.2.6 removeParticle()	19
3.3.2.7 step()	19
3.3.2.8 write()	19
3.4 Vector Struct Reference	20
3.4.1 Detailed Description	21
3.4.2 Constructor & Destructor Documentation	21
3.4.2.1 Vector()	21
3.4.3 Member Function Documentation	21
3.4.3.1 operator*()	21
3.4.3.2 operator+()	22
3.4.3.3 operator-()	22
3.4.3.4 operator/()	22
3.4.3.5 operator==()	23
3.4.3.6 size()	23
4 File Documentation	25
4.1 Array.hpp	25
4.2 constant.h	27
4.3 memtrace.h	27
4.4 Particle.hpp	30
4.5 Simulation.hpp	31
4.6 Vector.h	32
Index	33

Chapter 1

Class Index

1.1 Class List

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

Array< T >	Generic dynamic array class template	5
Particle	Non-interacting particle class	11
Simulation	Wrapper and manager for all simulation entities (currently only particle array)	16
Vector	Physics type vector3d	20

Chapter 2

File Index

2.1 File List

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

inc/ Array.hpp	25
inc/ constant.h	27
inc/ memtrace.h	27
inc/ Particle.hpp	30
inc/ Simulation.hpp	31
inc/ Vector.h	32

Chapter 3

Class Documentation

3.1 `Array< T >` Class Template Reference

Generic dynamic array class template.

```
#include <Array.hpp>
```

Collaboration diagram for Array< T >:

Array< T >
<ul style="list-style-type: none"> + Array(size_t cap=16) + Array(const Array &other) + Array & operator=(const Array &other) + ~Array() + T & operator[](size_t idx) + const T & operator[](size_t idx) const + bool operator==(const Array< T > &other) const + void print(std::ostream &os, const char *sep) + void insert(T *pelem, size_t idx) + void insert(T *pelem) + void remove(size_t idx) + void remove() + size_t getSize() const + size_t getCapacity() const + void write(std::ostream &os) const + void read(std::istream &is)

Public Member Functions

- [Array](#) (size_t cap=16)
Constructor.
- [Array](#) (const [Array](#) &other)
Copy constructor.
- [Array](#) & [operator=](#) (const [Array](#) &other)
Copy assignment.
- [~Array](#) ()
Destructor.

- T & `operator[]` (size_t idx)
Element access (non-const)
- const T & `operator[]` (size_t idx) const
Element access (const)
- bool `operator==` (const Array< T > &other) const
element by element comparison
- void `print` (std::ostream &os, const char *sep)
- void `insert` (T *pelem, size_t idx)
Insert an element at a specific index.
- void `insert` (T *pelem)
Insert an element at the end of the array.
- void `remove` (size_t idx)
Remove element at a specific index.
- void `remove` ()
Remove the last element.
- size_t `getSize` () const
Get the current size.
- size_t `getCapacity` () const
Get the current capacity.
- void `write` (std::ostream &os) const
array write to ostream
- void `read` (std::istream &is)
array read from istream

3.1.1 Detailed Description

```
template<typename T>
class Array< T >
```

Generic dynamic array class template.

Template Parameters

<i>T</i>	Type of elements stored
----------	-------------------------

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Array() [1/2]

```
template<typename T>
Array< T >::Array (
    size_t cap = 16) [inline]
```

Constructor.

Parameters

<i>cap</i>	Initial capacity
------------	------------------

3.1.2.2 Array() [2/2]

```
template<typename T>
Array< T >::Array (
    const Array< T > & other) [inline]
```

Copy constructor.

Parameters

<i>other</i>	Array to copy
--------------	---------------

3.1.3 Member Function Documentation

3.1.3.1 getCapacity()

```
template<typename T>
size_t Array< T >::getCapacity () const [inline]
```

Get the current capacity.

Returns

size_t Allocated capacity

3.1.3.2 getSize()

```
template<typename T>
size_t Array< T >::getSize () const [inline]
```

Get the current size.

Returns

size_t Number of elements

3.1.3.3 insert() [1/2]

```
template<typename T>
void Array< T >::insert (
    T * pelem) [inline]
```

Insert an element at the end of the array.

Parameters

<i>pelem</i>	Element to insert
--------------	-------------------

3.1.3.4 insert() [2/2]

```
template<typename T>
void Array< T >::insert (
    T * pelem,
    size_t idx) [inline]
```

Insert an element at a specific index.

Exceptions

<code>std::out_of_range</code>	when index is outside used array
--------------------------------	----------------------------------

Parameters

<i>pelem</i>	allocated element pointer (usage: <code>arr.insert(new T);</code>)
<i>idx</i>	Index at which to insert

3.1.3.5 operator=()

```
template<typename T>
Array & Array< T >::operator= (
    const Array< T > & other) [inline]
```

Copy assignment.

Parameters

<i>other</i>	Array to copy
--------------	-------------------------------

Returns

[Array](#)& Reference to this

3.1.3.6 operator==()

```
template<typename T>
bool Array< T >::operator== (
    const Array< T > & other) const [inline]
```

element by element comparison

Parameters

<i>other</i>	array to compare
--------------	------------------

Returns

bool

3.1.3.7 operator[]() [1/2]

```
template<typename T>
T & Array< T >::operator[] (
    size_t idx) [inline]
```

Element access (non-const)

Exceptions

<code>std::out_of_range</code>	when index is outside used array
--------------------------------	----------------------------------

Parameters

<code>idx</code>	Index
------------------	-------

Returns

T& Reference to element

3.1.3.8 operator[]() [2/2]

```
template<typename T>
const T & Array< T >::operator[] (
    size_t idx) const [inline]
```

Element access (const)

Exceptions

<code>std::out_of_range</code>	when index is outside used array
--------------------------------	----------------------------------

Parameters

<code>idx</code>	Index
------------------	-------

Returns

const T& Const reference to element

3.1.3.9 read()

```
template<typename T>
void Array< T >::read (
    std::istream & is) [inline]
```

array read from istream

Parameters

<code>is</code>	
-----------------	--

3.1.3.10 remove()

```
template<typename T>
void Array< T >::remove (
    size_t idx) [inline]
```

Remove element at a specific index.

Exceptions

<code>std::out_of_range</code>	when index is outside used array
--------------------------------	----------------------------------

Parameters

<code>idx</code>	Index to remove
------------------	-----------------

3.1.3.11 write()

```
template<typename T>
void Array< T >::write (
    std::ostream & os) const [inline]
```

array write to ostream

Parameters

<code>os</code>	ostream to write to
-----------------	---------------------

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

- inc/Array.hpp

3.2 Particle Class Reference

non-interacting particle class.

```
#include <Particle.hpp>
```

Collaboration diagram for Particle:

Particle
<ul style="list-style-type: none"> + Particle(Vector position=Vector(0, 0, 0), Vector velocity=Vector(0, 0, 0), double m=1, double ch=0, bool grav=false) + Particle(const Particle &) + virtual ~Particle() + Particle & operator=(const Particle &) + double getMass() const + double getCharge() const + bool getGrav() const + Vector getPos() const + Vector getVel() const + bool operator==(const Particle &other) const + bool operator!=(const Particle &other) const + Vector forceWith(const Particle &other) const + void applyForce(Vector force, double time) + void move(double time) + void write(std::ostream &os) const + void read(std::istream &is)

Public Member Functions

- **Particle** (Vector position=Vector(0, 0, 0), Vector velocity=Vector(0, 0, 0), double m=1, double ch=0, bool grav=false)
constructor
- **Particle** (const Particle &)
copy constructor
- virtual ~**Particle** ()
virtual destructor

- **Particle** & **operator=** (const **Particle** &)
copy assignment
- double **getMass** () const
mass getter
- double **getCharge** () const
charge getter
- bool **getGrav** () const
grav getter
- **Vector** **getPos** () const
position getter
- **Vector** **getVel** () const
velocity getter
- bool **operator==** (const **Particle** &other) const
equality operator
- bool **operator!=** (const **Particle** &other) const
- **Vector** **forceWith** (const **Particle** &other) const
calculates force between two particles.
- void **applyForce** (**Vector** force, double time)
applies force to particle
- void **move** (double time)
move the particle based on its velocity vector
- void **write** (std::ostream &os) const
write to ostream
- void **read** (std::istream &is)
read from ostream

3.2.1 Detailed Description

non-interacting particle class.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Particle()

```
Particle::Particle (
    Vector position = Vector(0,0,0),
    Vector velocity = Vector(0,0,0),
    double m = 1,
    double ch = 0,
    bool grav = false)
```

constructor

Parameters

<i>position</i>	position of particle
<i>velocity</i>	velocity of particle
<i>m</i>	mass of particle

3.2.3 Member Function Documentation

3.2.3.1 applyForce()

```
void Particle::applyForce (  
    Vector force,  
    double time)
```

applies force to particle

Parameters

<i>force</i>	force to apply
<i>time</i>	time to apply force for

3.2.3.2 forceWith()

```
Vector Particle::forceWith (  
    const Particle & other) const
```

calculates force between two particles.

Parameters

<i>other</i>	other particle
--------------	----------------

Returns

Vector force vector

3.2.3.3 getCharge()

```
double Particle::getCharge () const [inline]
```

charge getter

Returns

double charge

3.2.3.4 getGrav()

```
bool Particle::getGrav () const [inline]
```

grav getter

Returns

grav

3.2.3.5 getMass()

```
double Particle::getMass () const [inline]
```

mass getter

Returns

double mass attribute

3.2.3.6 getPos()

```
Vector Particle::getPos () const [inline]
```

position getter

Returns

[Vector](#) position attribute

3.2.3.7 getVel()

```
Vector Particle::getVel () const [inline]
```

velocity getter

Returns

[Vector](#) vel attribute

3.2.3.8 move()

```
void Particle::move (  
    double time)
```

move the particle based on its velocity vector

Parameters

<i>time</i>	how much time to move the particle for
-------------	--

3.2.3.9 operator==()

```
bool Particle::operator== (  
    const Particle & other) const
```

equality operator

Parameters

<i>other</i>	vector to compare
--------------	-------------------

3.2.3.10 read()

```
void Particle::read (  
    std::istream & is)
```

read from ostream

Parameters

<i>is</i>	std::istream to read from
-----------	---------------------------

3.2.3.11 write()

```
void Particle::write (  
    std::ostream & os) const
```

write to ostream

Parameters

<i>os</i>	std::ostream to write to
-----------	--------------------------

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

- inc/Particle.hpp
- src/Particle.cpp

3.3 Simulation Class Reference

wrapper and manager for all simulation entities (currently only particle array)

```
#include <Simulation.hpp>
```

Collaboration diagram for Simulation:

Simulation
<ul style="list-style-type: none"> + Simulation() + const Array< Particle > getParticles() const + bool operator==(const Simulation &other) const + void listParticles (std::ostream &os) + void addParticle(Particle *p) + void removeParticle (size_t idx) + void step(double t) + void write(std::ostream &os) const + void read(std::istream &is)

Public Member Functions

- **Simulation ()**
simulation constructor
- const **Array< Particle > getParticles ()** const
Get the Particles object (const)
- bool **operator==** (const **Simulation** &other) const
comparison
- void **listParticles** (std::ostream &os)
list particles to std::ostream
- void **addParticle** (**Particle** *p)
add a particle to simulation
- void **removeParticle** (size_t idx)
remove a particle from simulation
- void **step** (double t)
steps the simulation a set time
- void **write** (std::ostream &os) const
write sim to ostream
- void **read** (std::istream &is)
read sim from istream

3.3.1 Detailed Description

wrapper and manager for all simulation entities (currently only particle array)

3.3.2 Member Function Documentation

3.3.2.1 addParticle()

```
void Simulation::addParticle (
    Particle * p)
```

add a particle to simulation

Parameters

p	particle to add
-----	-----------------

3.3.2.2 getParticles()

```
const Array< Particle > Simulation::getParticles () const [inline]
```

Get the Particles object (const)

Returns

const [Array<Particle>](#)

3.3.2.3 listParticles()

```
void Simulation::listParticles (
    std::ostream & os)
```

list particles to std::ostream

Parameters

os	std::ostream to list
------	----------------------

3.3.2.4 operator==()

```
bool Simulation::operator== (
    const Simulation & other) const
```

comparison

Parameters

<i>other</i>	other simulation
--------------	------------------

3.3.2.5 read()

```
void Simulation::read (  
    std::istream & is)
```

read sim from istream

Parameters

<i>is</i>	istream
-----------	---------

3.3.2.6 removeParticle()

```
void Simulation::removeParticle (  
    size_t idx)
```

remove a particle from simulation

Parameters

<i>idx</i>	index of particle to remove
------------	-----------------------------

3.3.2.7 step()

```
void Simulation::step (  
    double t)
```

steps the simulation a set time

Parameters

<i>t</i>	time to step
----------	--------------

3.3.2.8 write()

```
void Simulation::write (  
    std::ostream & os) const
```

write sim to ostream

Parameters

os	ostream
----	---------

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

- inc/Simulation.hpp
- src/Simulation.cpp

3.4 Vector Struct Reference

physics type vector3d

```
#include <Vector.h>
```

Collaboration diagram for Vector:

Vector
+ double x
+ double y
+ double z
+ Vector(double X=0, double Y=0, double Z=0)
+ double size()
+ bool operator==(const Vector other) const
+ Vector operator+(const Vector other) const
+ Vector operator-(const Vector other) const
+ Vector operator*(const double scalar) const
+ Vector operator/(const double scalar) const

Public Member Functions

- `Vector` (double X=0, double Y=0, double Z=0)
vector constructor
- double `size` ()
norm of vector (pythagoras)
- bool `operator==` (const `Vector` other) const
field by field equality
- `Vector operator+` (const `Vector` other) const
vector addition
- `Vector operator-` (const `Vector` other) const
vector subtraction
- `Vector operator*` (const double scalar) const
scalar multiplication
- `Vector operator/` (const double scalar) const
scalar division

Public Attributes

- double `x`
- double `y`
- double `z`

3.4.1 Detailed Description

physics type vector3d

3.4.2 Constructor & Destructor Documentation

3.4.2.1 Vector()

```
Vector::Vector (
    double X = 0,
    double Y = 0,
    double Z = 0)
```

vector constructor

Parameters

X	x coordinate
Y	y coordinate
Z	z coordinate

3.4.3 Member Function Documentation

3.4.3.1 operator*()

```
Vector Vector::operator* (
    const double scalar) const
```

scalar multiplication

Parameters

<i>scalar</i>	scalar to multiply vector by
---------------	------------------------------

Returns

[Vector](#) scaled vector

3.4.3.2 operator+()

```
Vector Vector::operator+ (  
    const Vector other) const
```

vector addition

Parameters

<i>other</i>	vector to sum with
--------------	--------------------

Returns

[Vector](#) sum of vectors

3.4.3.3 operator-()

```
Vector Vector::operator- (  
    const Vector other) const
```

vector subtraction

Parameters

<i>vector</i>	to subtract
---------------	-------------

Returns

[Vector](#) difference of vectors

3.4.3.4 operator/()

```
Vector Vector::operator/ (  
    const double scalar) const
```

scalar division

Parameters

<i>scalar</i>	scalar to divide by
---------------	---------------------

Returns

[Vector](#) scaled vector

3.4.3.5 operator==()

```
bool Vector::operator== (
    const Vector other) const
```

field by field equality

Parameters

<i>other</i>	vector to compare
--------------	-------------------

Returns

bool

3.4.3.6 size()

```
double Vector::size ()
```

norm of vector (pythagoras)

Returns

double norm

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

- inc/Vector.h
- src/Vector.cpp

Chapter 4

File Documentation

4.1 Array.hpp

```
00001 #ifndef ARRAY_HPP
00002 #define ARRAY_HPP
00003
00004 #include <cstdint>
00005 #include <iostream>
00006 #include <stdexcept>
00007
00008 #include "memtrace.h"
00009
00015 template<typename T>
00016 class Array {
00017 private:
00018     T** data;
00019     size_t size;
00020     size_t capacity;
00021
00027     void deepCopy(const Array& other)
00028     {
00029         data = new T*[capacity];
00030         for (size_t i = 0; i < size; ++i)
00031         {
00032             data[i] = new T(*other.data[i]);
00033         }
00034     }
00035
00041     void doubleCapacity()
00042     {
00043         if (size < capacity) throw std::length_error("Too few elements to double capacity");
00044         T** temp = new T*[capacity * 2];
00045
00046         for(size_t i = 0; i < size; i++)
00047             temp[i] = data[i];
00048
00049         delete[] data;
00050         capacity *= 2;
00051         data = temp;
00052     }
00053
00059     void halveCapacity()
00060     {
00061         if (size > capacity / 2) throw std::length_error("Too many elements to halve capacity");
00062         if (capacity <= 2) return;
00063
00064         T** temp = new T*[capacity / 2];
00065
00066         for(size_t i = 0; i < size; i++)
00067             temp[i] = data[i];
00068
00069         delete[] data;
00070         data = temp;
00071     }
00072
00073 public:
00080     Array(size_t cap = 16)
00081         : data(nullptr), size(0), capacity(cap)
00082     {
```

```

00083     data = new T*[capacity];
00084 }
00085
00091 Array(const Array& other)
00092 : data(nullptr), size(other.size), capacity(other.capacity)
00093 {
00094     deepCopy(other);
00095 }
00096
00103 Array& operator=(const Array& other)
00104 {
00105     if (this == &other) return *this;
00106
00107     for (size_t i = 0; i < size; i++)
00108         delete data[i];
00109     delete[] data;
00110
00111     size = other.size;
00112     capacity = other.capacity;
00113     deepCopy(other);
00114
00115     return *this;
00116 }
00117
00121 ~Array()
00122 {
00123     for (size_t i = 0; i < size; i++)
00124         delete data[i];
00125     delete[] data;
00126 }
00127
00135 T& operator[](size_t idx)
00136 {
00137     if (idx >= size) throw std::out_of_range("Index out of size");
00138     return *data[idx];
00139 }
00140
00148 const T& operator[](size_t idx) const
00149 {
00150     if (idx >= size) throw std::out_of_range("Index out of size");
00151     return *data[idx];
00152 }
00153
00160 bool operator==(const Array<T>& other) const
00161 {
00162     if (size != other.size) return false;
00163
00164     for (size_t i = 0; i < size; i++)
00165         if (*data[i] != *other.data[i]) return false;
00166
00167     return true;
00168 }
00169
00170 void print(std::ostream& os, const char* sep)
00171 {
00172     for (size_t i = 0; i < size-1; i++)
00173         os << *data[i] << sep;
00174     os << *data[size-1];
00175 }
00176
00184 void insert(T* pelem, size_t idx)
00185 {
00186     if (idx > size) throw std::out_of_range("Insert index out of bounds");
00187     if (size >= capacity)
00188         doubleCapacity();
00189
00190     for (size_t i = size; i > idx; i--) {
00191         *data[i] = *data[i - 1];
00192     }
00193
00194     data[idx] = pelem;
00195     size++;
00196 }
00197
00203 void insert(T* pelem)
00204 {
00205     insert(pelem, size);
00206 }
00207
00214 void remove(size_t idx)
00215 {
00216     if (idx >= size) throw std::out_of_range("Remove index out of bounds");
00217     if (size < capacity/2) halveCapacity();
00218
00219     delete data[idx];
00220
00221     for (size_t i = idx; i < size - 1; ++i) {

```

```

00222         data[i] = data[i + 1];
00223     }
00224
00225     size--;
00226 }
00227
00231 void remove()
00232 {
00233     remove(size - 1);
00234 }
00235
00241 size_t getSize() const
00242 {
00243     return size;
00244 }
00245
00251 size_t getCapacity() const
00252 {
00253     return capacity;
00254 }
00255
00261 void write(std::ostream& os) const
00262 {
00263     os << size << std::endl;
00264     for(size_t i = 0; i < size; i++)
00265     {
00266         data[i]->write(os);
00267         os << std::endl;
00268     }
00269 }
00270
00276 void read(std::istream& is)
00277 {
00278     for (size_t i = 0; i < size; i++)
00279         delete data[i];
00280
00281     (is >> size).ignore(1);
00282     for(size_t i = 0; i < size; i++)
00283     {
00284         data[i] = new T;
00285         data[i]->read(is);
00286         is.ignore(1);
00287     }
00288 }
00289
00290 };
00291
00300 template<typename T>
00301 std::ostream& operator<<(std::ostream& os, const Array<T>& arr)
00302 {
00303     for (size_t i = 0; i < arr.getSize(); i++)
00304     {
00305         os << " index :" << i << std::endl;
00306         os << arr[i];
00307     }
00308     return os;
00309 }
00310
00311 #endif // ARRAY_HPP

```

4.2 constant.h

```

00001 #ifndef CONSTANT_H
00002 #define CONSTANT_H
00003
00004 #define EPSILON 0.0001
00005 #define G 1
00006 #define K 1
00007
00008 #endif

```

4.3 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 include-ok:   2017., 2018., 2019., 2021., 2022.
00009 clang-mágia:Bodor András, 2025
00010 *****/
00011
00012 #ifndef MEMTRACE_H
00013 #define MEMTRACE_H
00014
00015 #if defined(MEMTRACE)
00016
00017 /*ha definiálva van, akkor a hibákat ebbe a fájlba írja, egyébként stderr-re*/
00018 /*#define MEMTRACE_ERRFILE MEMTRACE.ERR*/
00019
00020 /*ha definiálva van, akkor futás közben lancolt listát épít. Javasolt a használata*/
00021 #define MEMTRACE_TO_MEMORY
00022
00023 /*ha definiálva van, akkor futás közben fájlba írja a foglalásokat*/
00024 /*ekkor nincs ellenőrzés, csak naplózás*/
00025 /*#define MEMTRACE_TO_FILE*/
00026
00027 /*ha definiálva van, akkor a megállaskor automatikus riport készül */
00028 #define MEMTRACE_AUTO
00029
00030 /*ha definiálva van, akkor malloc()/calloc()/realloc()/free() követve lesz*/
00031 #define MEMTRACE_C
00032
00033 #ifdef MEMTRACE_C
00034     /*ha definiálva van, akkor free(NULL) nem okoz hibát*/
00035     #define ALLOW_FREE_NULL
00036 #endif
00037
00038 #ifdef __cplusplus
00039     /*ha definiálva van, akkor new/delete/new[]/delete[] követve lesz*/
00040     #define MEMTRACE_CPP
00041 #endif
00042
00043 #if defined(__cplusplus) && defined(MEMTRACE_TO_MEMORY)
00044     /*ha definiálva van, akkor atexit helyett objektumot használ*/
00045     /*ajánlott bekapcsolni*/
00046     #define USE_ATEXIT_OBJECT
00047 #endif
00048
00049 /*****/
00050 /* INNEN NE MODOSÍTSD */
00051 /*****/
00052 #ifndef NO_MEMTRACE_TO_FILE
00053     #undef MEMTRACE_TO_FILE
00054 #endif
00055
00056 #ifndef NO_MEMTRACE_TO_MEMORY
00057     #undef MEMTRACE_TO_MEMORY
00058 #endif
00059
00060 #ifndef MEMTRACE_AUTO
00061     #undef USE_ATEXIT_OBJECT
00062 #endif
00063
00064 #ifdef __cplusplus
00065     #define START_NAMESPACE namespace memtrace {
00066     #define END_NAMESPACE } /*namespace*/
00067     #define TRACEC(func) memtrace::func
00068     #include <new>
00069 #else
00070     #define START_NAMESPACE
00071     #define END_NAMESPACE
00072     #define TRACEC(func) func
00073 #endif
00074
00075 // THROW deklaráció változatai
00076 #if defined(_MSC_VER)
00077     // VS rosszul kezeli az __cplusplus makrot
00078     #if _MSC_VER < 1900
00079         // * nem biztos, hogy jó így *
00080         #define THROW_BADALLOC
00081         #define THROW_NOTHING
00082     #else
00083         // C++11 vagy újabb
00084         #define THROW_BADALLOC noexcept(false)
00085         #define THROW_NOTHING noexcept
00086     #endif
00087 #else
00088     #if __cplusplus < 201103L
00089         // C++2003 vagy régebbi
00090         #define THROW_BADALLOC throw (std::bad_alloc)
00091         #define THROW_NOTHING throw ()
00092     #else

```

```

00093     // C++11 vagy újabb
00094     #define THROW_BADALLOC noexcept(false)
00095     #define THROW_NOTHING noexcept
00096 #endif
00097 #endif
00098
00099 START_NAMESPACE
00100     int allocated_blocks();
00101 END_NAMESPACE
00102
00103 #if defined(MEMTRACE_TO_MEMORY)
00104 START_NAMESPACE
00105     int mem_check(void);
00106     int poi_check(void*);
00107 END_NAMESPACE
00108 #endif
00109
00110 #if defined(MEMTRACE_TO_MEMORY) && defined(USE_ATEXIT_OBJECT)
00111 #include <cstdio>
00112 START_NAMESPACE
00113     class atexit_class {
00114     private:
00115         static int counter;
00116         static int err;
00117     public:
00118         atexit_class() {
00119 #if defined(CPORTA) && !defined(CPORTA_NOSETBUF)
00120             if (counter == 0) {
00121                 setbuf(stdout, 0);
00122                 setbuf(stderr, 0);
00123             }
00124 #endif
00125             counter++;
00126         }
00127
00128         int check() {
00129             if(--counter == 0)
00130                 err = mem_check();
00131             return err;
00132         }
00133
00134         ~atexit_class() {
00135             check();
00136         }
00137     };
00138
00139     static atexit_class atexit_obj;
00140
00141 END_NAMESPACE
00142 #endif /* MEMTRACE_TO_MEMORY && USE_ATEXIT_OBJECT */
00143
00144 /* Innentől csak a "normal" include eseten kell, különben összezavarja a mukodest */
00145 #ifndef FROM_MEMTRACE_CPP
00146 #include <stdlib.h>
00147 #ifdef __cplusplus
00148     #include <iostream>
00149     /* ide gyűjtjük a memtrace-vel összeakadó headereket, hogy előbb legyenek */
00150
00151     #include <fstream> // VS 2013 headerjében van deleted definíció
00152     #include <sstream>
00153     #include <vector>
00154     #include <list>
00155     #include <map>
00156     #include <algorithm>
00157     #include <functional>
00158     #include <memory>
00159     #include <iomanip>
00160     #include <locale>
00161     #include <typeinfo>
00162     #include <ostream>
00163     #include <stdexcept>
00164     #include <ctime>
00165     #include <random>
00166     #if __cplusplus >= 201103L
00167         #include <iterator>
00168         #include <regex>
00169     #endif
00170 #endif
00171 #ifdef MEMTRACE_CPP
00172     namespace std {
00173         typedef void (*new_handler)();
00174     }
00175 #endif
00176
00177 #ifdef MEMTRACE_C
00178 START_NAMESPACE
00179     #undef malloc

```

```

00180     #define malloc(size) TRACEC(traced_malloc)(size, #size, __LINE__, __FILE__)
00181     void * traced_malloc(size_t size, const char *size_txt, int line, const char * file);
00182
00183     #undef calloc
00184     #define calloc(count, size) TRACEC(traced_calloc)(count, size, #count, "#size, __LINE__, __FILE__")
00185     void * traced_calloc(size_t count, size_t size, const char *size_txt, int line, const char *
file);
00186
00187     #undef free
00188     #define free(p) TRACEC(traced_free)(p, #p, __LINE__, __FILE__)
00189     void traced_free(void * p, const char *size_txt, int line, const char * file);
00190
00191     #undef realloc
00192     #define realloc(old, size) TRACEC(traced_realloc)(old, size, #size, __LINE__, __FILE__)
00193     void * traced_realloc(void * old, size_t size, const char *size_txt, int line, const char * file);
00194
00195     void mem_dump(void const *mem, size_t size, FILE* fp = stdout);
00196
00197 END_NAMESPACE
00198 #endif /* MEMTRACE_C */
00199
00200 #ifdef MEMTRACE_CPP
00201 START_NAMESPACE
00202     #undef set_new_handler
00203     #define set_new_handler(f) TRACEC(_set_new_handler)(f)
00204     void _set_new_handler(std::new_handler h);
00205
00206     void set_delete_call(int line, const char * file);
00207 END_NAMESPACE
00208
00209 void * operator new(size_t size, int line, const char * file) THROW_BADALLOC;
00210 void * operator new[](size_t size, int line, const char * file) THROW_BADALLOC;
00211 void * operator new(size_t size) THROW_BADALLOC;
00212 void * operator new[](size_t size) THROW_BADALLOC;
00213 void operator delete(void * p) THROW_NOTHING;
00214 void operator delete[](void * p) THROW_NOTHING;
00215
00216 #if __cplusplus >= 201402L
00217 // sized delete miatt: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3536.html
00218 void operator delete(void * p, size_t) THROW_NOTHING;
00219 void operator delete[](void * p, size_t) THROW_NOTHING;
00220 #endif
00221
00222 /* Visual C++ 2012 miatt kell, mert háklis, hogy nincs megfelelő delete, bár senki sem használja */
00223 void operator delete(void *p, int, const char *) THROW_NOTHING;
00224 void operator delete[](void *p, int, const char *) THROW_NOTHING;
00225
00226 // clang >= 3.1 esetén vannak warningok, amiket zavar, hogy redefiniálva van a new/delete
00227 #if defined(__clang__) && (__clang_major__ > 3 || \
00228     (__clang_major__ == 3 && __clang_minor__ > 0))
00229     // Csak nagyon drasztikus warning szint mellett jön elő, amikor van rekurzívnek tűnő makró.
00230     // Ilyenek a new és delete alább, hiszen olyan, mintha magukat hívnák, pedig nincs (ilyen)
00231     // rekurzió makró szinten.
00232     #pragma clang diagnostic ignored "-Wdisabled-macro-expansion"
00233     // Bármilyen kulcsszó újradefiniálása esetén pánikol.
00234     #pragma clang diagnostic ignored "-Wkeyword-macro"
00235 #endif
00236
00237 #define new new(__LINE__, __FILE__)
00238 #define delete memtrace::set_delete_call(__LINE__, __FILE__), delete
00239
00240 #ifdef CPORTA
00241 #define system(...) // system(__VA_ARGS__)
00242 #endif
00243
00244 #endif /* MEMTRACE_CPP */
00245
00246 #endif /* FROM_MEMTRACE_CPP */
00247 #else
00248 #pragma message ( "MEMTRACE NOT DEFINED" )
00249 #endif /* MEMTRACE */
00250
00251 #endif /* MEMTRACE_H */

```

4.4 Particle.hpp

```

00001 #ifndef PARTICLE_HPP
00002 #define PARTICLE_HPP
00003
00004 #include "Vector.h"
00005 #include <ostream>
00006
00011 class Particle

```

```

00012 {
00013     Vector pos;
00014     Vector vel;
00015     double mass;
00016     double charge;
00017     bool isGravityInteracting;
00018
00019 public:
00020
00028     Particle(Vector position = Vector(0,0,0),
00029             Vector velocity = Vector(0,0,0),
00030             double m = 1,
00031             double ch = 0,
00032             bool grav = false);
00033
00037     Particle(const Particle&);
00038
00042     virtual ~Particle();
00043
00047     Particle& operator=(const Particle&);
00048
00054     double getMass() const { return mass; }
00055
00061     double getCharge() const { return charge; }
00062
00068     bool getGrav() const { return isGravityInteracting; }
00069
00075     Vector getPos() const { return pos; }
00076
00082     Vector getVel() const { return vel; }
00083
00088     bool operator==(const Particle& other) const;
00089
00090     bool operator!=(const Particle& other) const { return !operator==(other); }
00091
00098     Vector forceWith(const Particle& other) const;
00099
00106     void applyForce(Vector force, double time);
00107
00113     void move(double time);
00114
00120     void write(std::ostream& os) const;
00121
00127     void read(std::istream& is);
00128
00129 };
00130
00138 std::ostream& operator<<(std::ostream& os, const Particle& p);
00139
00140 #endif

```

4.5 Simulation.hpp

```

00001 #ifndef SIMULATION_HPP
00002 #define SIMULATION_HPP
00003
00004 #include "Array.hpp"
00005 #include "Particle.hpp"
00006
00011 class Simulation {
00012     Array<Particle> particles;
00013 public:
00014
00018     Simulation();
00019
00025     const Array<Particle> getParticles() const { return particles; }
00026
00032     bool operator==(const Simulation& other) const;
00033
00039     void listParticles(std::ostream& os);
00040
00046     void addParticle(Particle* p);
00047
00053     void removeParticle(size_t idx);
00054
00060     void step(double t);
00061
00066     void write(std::ostream& os) const;
00067
00073     void read(std::istream& is);
00074
00075 };
00076

```



```
00077 std::ostream& operator<<(std::ostream& os, const Simulation& sim);
00078
00079
00080 #endif
```

4.6 Vector.h

```
00001 #ifndef VECTOR_H
00002 #define VECTOR_H
00003
00004 #include <iostream>
00005
00006
00011 struct Vector {
00012     double x,y,z;
00013
00021     Vector(double X = 0, double Y = 0, double Z = 0);
00022
00028     double size();
00029
00036     bool operator==(const Vector other) const;
00037
00044     Vector operator+(const Vector other) const;
00045
00052     Vector operator-(const Vector other) const;
00053
00060     Vector operator*(const double scalar) const;
00061
00068     Vector operator/(const double scalar) const;
00069 };
00070
00071 //gtest
00072 std::ostream& operator<<(std::ostream& os, Vector v);
00073
00074
00075 #endif // VECTOR_H
```

Index

- addParticle
 - Simulation, 18
- applyForce
 - Particle, 14
- Array
 - Array< T >, 7
- Array< T >, 5
 - Array, 7
 - getCapacity, 8
 - getSize, 8
 - insert, 8
 - operator=, 9
 - operator==, 9
 - operator[], 9, 10
 - read, 10
 - remove, 10
 - write, 11
- forceWith
 - Particle, 14
- getCapacity
 - Array< T >, 8
- getCharge
 - Particle, 14
- getGrav
 - Particle, 14
- getMass
 - Particle, 14
- getParticles
 - Simulation, 18
- getPos
 - Particle, 15
- getSize
 - Array< T >, 8
- getVel
 - Particle, 15
- inc/Array.hpp, 25
- inc/constant.h, 27
- inc/memtrace.h, 27
- inc/Particle.hpp, 30
- inc/Simulation.hpp, 31
- inc/Vector.h, 32
- insert
 - Array< T >, 8
- listParticles
 - Simulation, 18
- move
 - Particle, 15
- operator+
 - Vector, 22
- operator-
 - Vector, 22
- operator/
 - Vector, 22
- operator=
 - Array< T >, 9
- operator==
 - Array< T >, 9
 - Particle, 15
 - Simulation, 18
 - Vector, 23
- operator[]
 - Array< T >, 9, 10
- operator*
 - Vector, 21
- Particle, 11
 - applyForce, 14
 - forceWith, 14
 - getCharge, 14
 - getGrav, 14
 - getMass, 14
 - getPos, 15
 - getVel, 15
 - move, 15
 - operator==, 15
 - Particle, 13
 - read, 16
 - write, 16
- read
 - Array< T >, 10
 - Particle, 16
 - Simulation, 19
- remove
 - Array< T >, 10
- removeParticle
 - Simulation, 19
- Simulation, 16
 - addParticle, 18
 - getParticles, 18
 - listParticles, 18
 - operator==, 18
 - read, 19
 - removeParticle, 19

- step, [19](#)
 - write, [19](#)
- size
 - Vector, [23](#)
- step
 - Simulation, [19](#)
- Vector, [20](#)
 - operator+, [22](#)
 - operator-, [22](#)
 - operator/, [22](#)
 - operator==, [23](#)
 - operator*, [21](#)
 - size, [23](#)
 - Vector, [21](#)
- write
 - Array< T >, [11](#)
 - Particle, [16](#)
 - Simulation, [19](#)