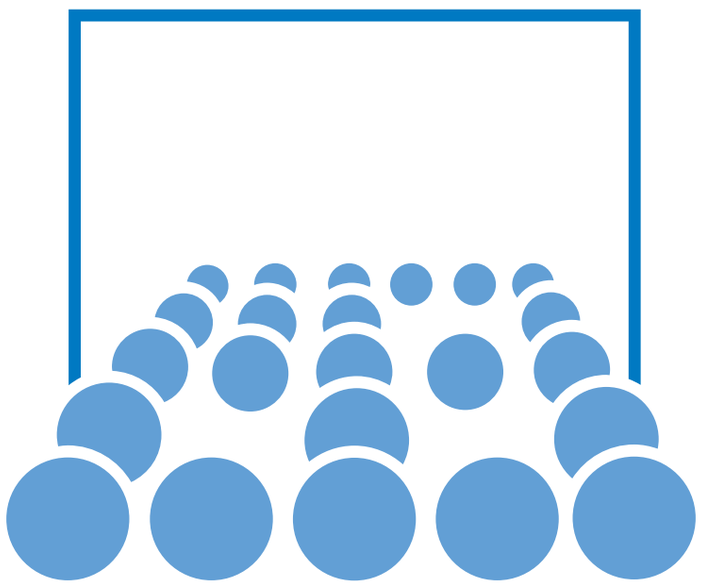
PSE Molekulardynamik

MD-basics,
Unit tests & Assertions

Alexander Breuer, Wolfgang Eckhardt

02.11.2012



Outline

- Schedule
- Presentations: Worksheet 1
- Lennard-Jones potential & Brownian motion
- Assertions & Unit tests
- Preparation: Worksheet 2

Schedule (big meetings)

Date	Worksheet
19.10.2012	
02.11.2012	1
16.11.2012	2
07.12.2012	3
21.12.2012	4
01.02.2012	5

Presentations: Worksheet 1

Institut für Informatik — TU München
Scientific **C**omputing in **C**omputer **S**cience
Prof. Dr. H.-J. Bungartz
Dipl.-Inf. W. Eckhardt
Dipl.-Math. A. Breuer

WS 2012/13

PSE Molekulardynamik
Sheet 1: First Steps towards a Moleculardynamics Simulation

Exercise on 19th October 2012.

General Information

- Work in groups of 2 or 3 students
- Deliverables:
  - GIT repository containing your source code, compilable with Makefile provided.
  - HTML-Documentation generated with Doxygen.
- Deliver latest by 3 p.m. of the given date per email.

Task 1 „First Steps “

- Download and install Eclipse CDT (http://www.eclipse.org/downloads/)
- Setup GIT (http://git-scm.com/)
- Download the programme frame from GitHub, compile and execute it.
- Download and install VTK/Paraview (http://paraview.org)
- Download and install Doxygen (http://www.doxygen.org). Create a preliminary documentation of the code frame.

Note:

- The directory libxsd/xsd has to be on the include path.
- You have to link against the Xerces XML Parser (flag -lxerces-c)

Task 2 „Completion of the programme frame “

- As we discussed in the lecture, the basic algorithm of the molecular dynamics simulation consists of the following steps:
  - Force calculation.
  - Calculation of the new position according to these forces.
  - Calculation of the new velocities according to these forces.
- Complete the steps of the simulation in the programme frame.
- Create VTK output for visualization with the VTKWriter class.
- Pass the parameters t_end and delta_t via the command line! (You can use the function atof for parsing the string to float)

Note:

- Visualize molecules in paraview with a *glyph*.
- It is possible to export videos (.avi) with “File“ → “save Animation“

Task 3 „Refactoring and Documentation “

The code frame in its current state is not very suitable as base for building up a moleculardynamics simulator. Refactor it, especially with respect to the following issues:

- Encapsulate the molecule list into a class ParticleContainer. It should be possible to iterate over the particles as well as the particle pairs in a simple way. Which design pattern(s) is / are suitable?
- During this course, we will use different methods for I/O and for calculating the force between particles. Think about an easy and extensible way to do that.
- Make yourself familiar with Doxygen. What can you do with a system like Doxygen, what is it useful for?

  Document the interfaces / implementations you created with Doxygen.

Good luck!

Deliver by 1st November 2012, 3 pm per mail to `breuera@in.tum.de`!

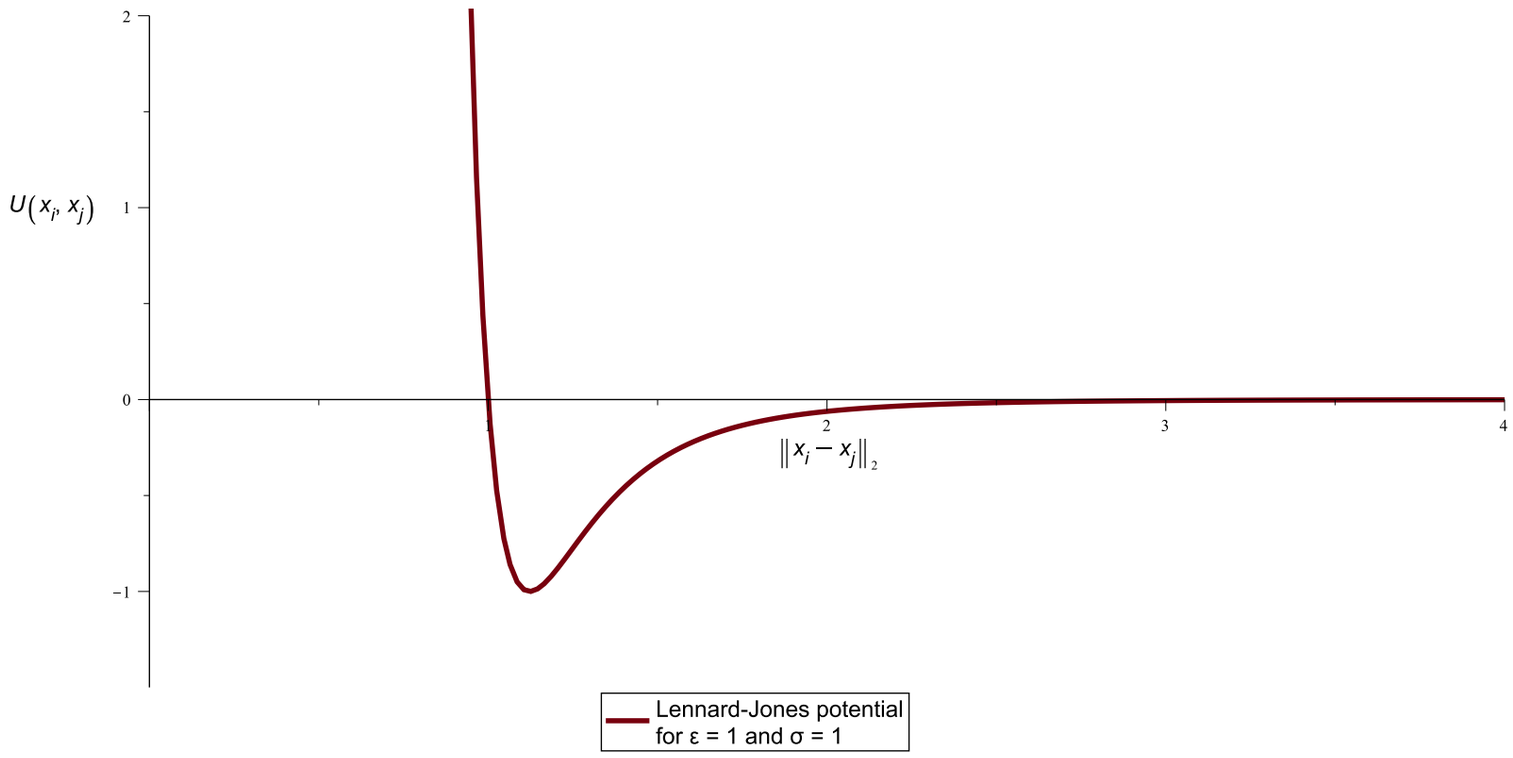
The programme frame is locatet at
`https://github.com/TUM-I5/MolSim`

Lennard-Jones potential

- Interaction between molecules or atoms

$$U(x_i, x_j) = 4\epsilon \left(\left(\frac{\sigma}{\|x_i - x_j\|_2} \right)^{12} - \left(\frac{\sigma}{\|x_i - x_j\|_2} \right)^{6} \right)$$

solidity



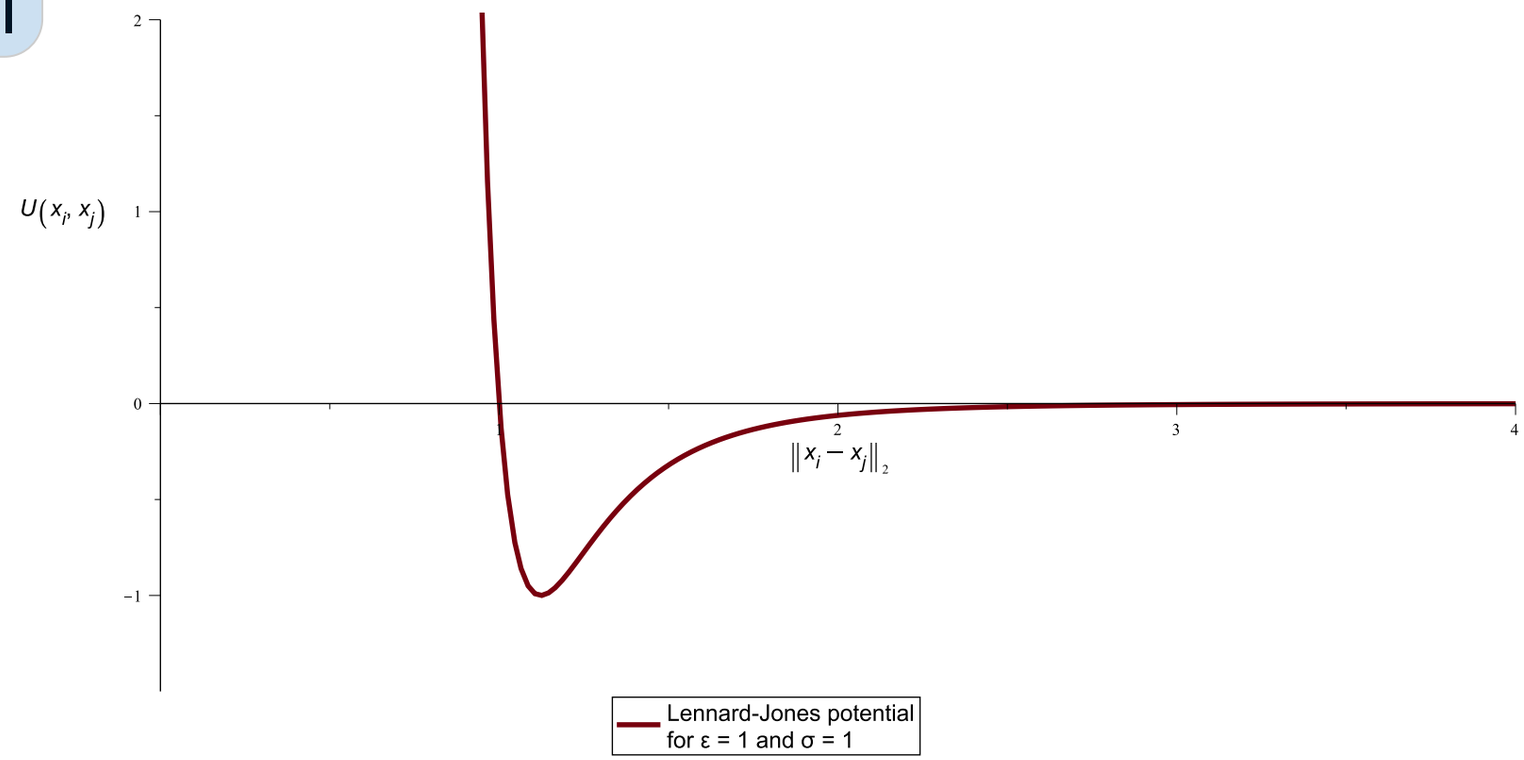
Lennard-Jones potential

- Interaction between molecules or atoms

$$U(x_i, x_j) = 4\epsilon \left(\left(\frac{\sigma}{\|x_i - x_j\|_2} \right)^{12} - \left(\frac{\sigma}{\|x_i - x_j\|_2} \right)^{6} \right)$$

potential well

solidity



Lennard-Jones potential

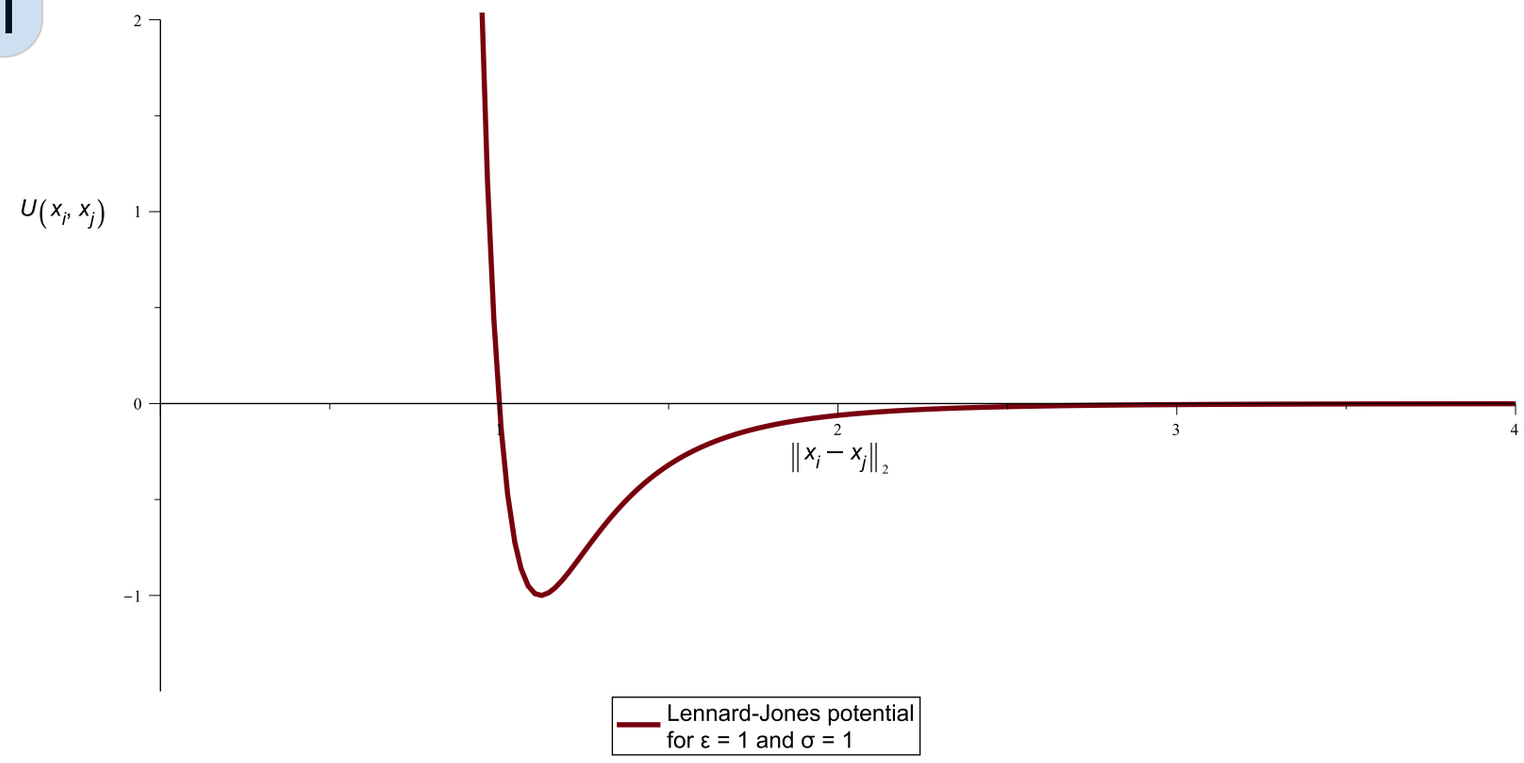
- Interaction between molecules or atoms

$$U(x_i, x_j) = 4\epsilon \left(\left(\frac{\sigma}{\|x_i - x_j\|_2} \right)^{12} - \left(\frac{\sigma}{\|x_i - x_j\|_2} \right)^{6} \right)$$

zero-crossing

potential well

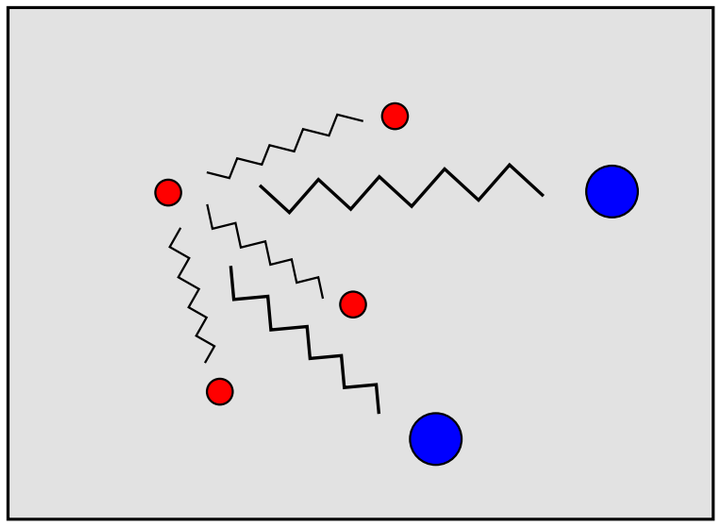
solidity



Lennard-Jones potential

- Resulting force calculation (sheet 2)

$$F_i = \sum_{\substack{j=1 \\ j \neq i}}^{\#particles} F_{ij}, \quad F_{ij} = \frac{24\epsilon}{(\|x_i - x_j\|_2)^2} \left(\left(\frac{\sigma}{\|x_i - x_j\|_2} \right)^6 - 2 \left(\frac{\sigma}{\|x_i - x_j\|_2} \right)^{12} \right) (x_j - x_i)$$



Brownian Motion

- Temperature ⇒ random movement of particles
- Implementation: Maxwell-Boltzmann distribution

$$f(v) = \left(\frac{m}{2\pi k_B T}\right)^{\frac{3}{2}} e^{-\frac{m v^2}{2 k_B T}}$$

- Sheet 2: Code @ homepage

Sheet 2: 2D-functionality of the function

Brownian Motion

- Temperature ⇒ random movement of particles
- Implementation: Maxwell-Boltzmann distribution

$$f(v) = \left(\frac{m}{2\pi k_B T}\right)^{\frac{3}{2}} e^{-\frac{m v^2}{2 k_B T}}$$

Boltzmann constant

- Sheet 2: Code @ homepage

Sheet 2: 2D-functionality of the function

Brownian Motion

- Temperature ⇒ random movement of particles
- Implementation: Maxwell-Boltzmann distribution

$$f(v) = \left(\frac{m}{2\pi k_B T}\right)^{\frac{3}{2}} e^{-\frac{m v^2}{2 k_B T}}$$

Boltzmann constant

temperature

- Sheet 2: Code @ homepage

Sheet 2: 2D-functionality of the function

Brownian Motion

- Temperature ⇒ random movement of particles
- Implementation: Maxwell-Boltzmann distribution

$$f(v) = \left(\frac{m}{2\pi k_B T}\right)^{\frac{3}{2}} e^{-\frac{m v^2}{2 k_B T}}$$

mass

Boltzmann constant

temperature

- Sheet 2: Code @ homepage

Sheet 2: 2D-functionality of the function

Hands on: Assertions

- Example: Hello world

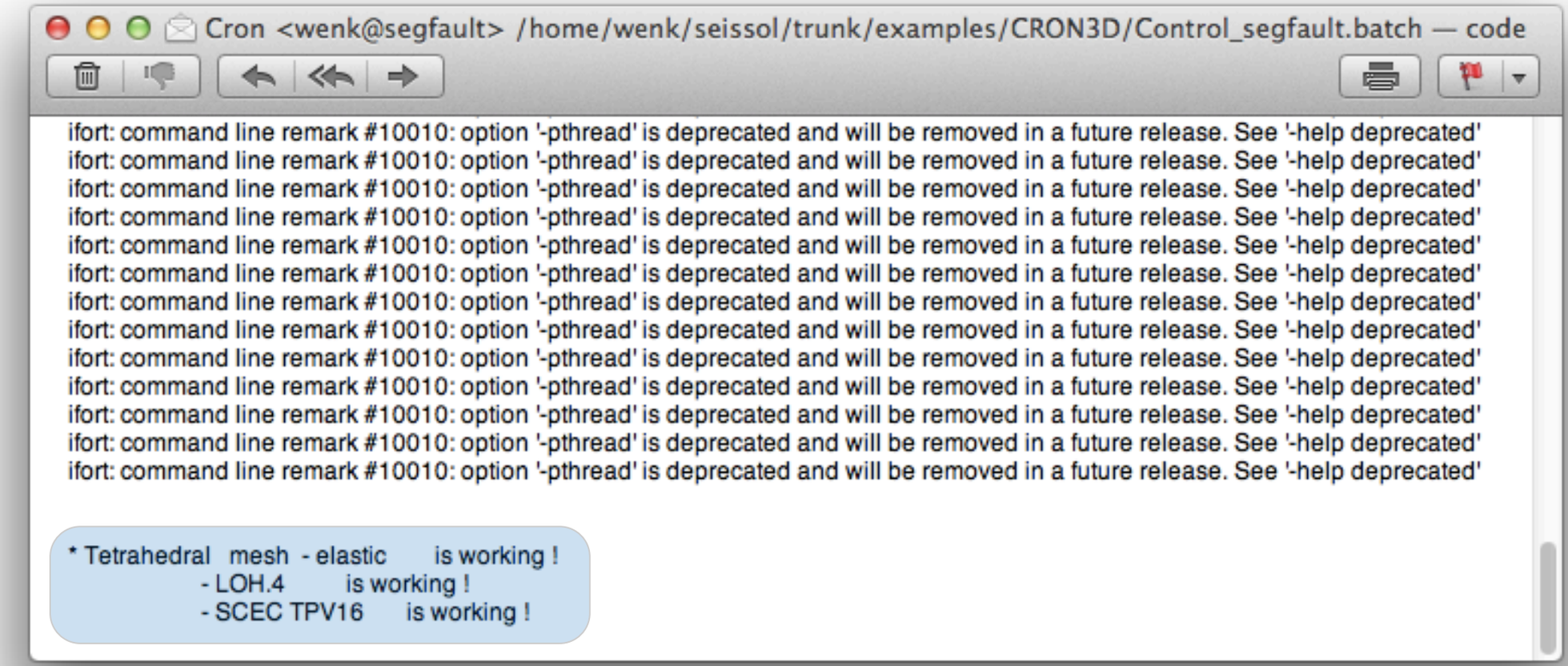
hello_assert.cpp

hello_assert.cpp › main()

```
1  #include <cassert>
2  #include <iostream>
3
4  int main() {
5    std::cout << "Hello World!" << std::endl;
6
7    double l_array[10];
8
9    for( int l_i = 0; l_i < 15; l_i++ ) {
10     assert(l_i < 10);
11
12     l_array[l_i] = (double) l_i;
13
14     std::cout << l_array[l_i] << std::endl;
15   }
16
17   return 0;
18 }
```

Unit test

- Check functionality of each program part
- Overnight builds common



Preparation: Worksheet 2

Institut für Informatik — TU München
Scientific **C**omputing in **C**omputer **S**cience
Prof. Dr. H.-J. Bungartz
Dipl.-Inf. W. Eckhardt
Dipl.-Math. A. Breuer

WS 2012/13

PSE Molekulardynamik
Sheet 2: Collision of two bodies

Exercise at 2nd November 2012

Task 1 „Unit Tests“

Basis for the further development of the simulator will be that it is possible to test the functionality of components before they are completely integrated with the existing code.

- Inform yourself about CppUnit and assertions in C++. Download CppUnit (http://apps.sourceforge.net/mediawiki/cppunit/).
- Build the CppUnit-library according to the instructions (file INSTALL).
- Write a simple UnitTest for the particle container you created in sheet 1, as described in the CppUnit CookBook. Also implement a runner. The unit tests should be executed when your programme is called with the option “–test”.
- Extend the code so that it is possible to specify the name of a single test case which is then being executed.
- In the following parts of the PSE always add assertions and new tests to the components you implement!

Task 2 „Logging “

For debugging it can be very useful to make use of a logging framework. We will use Log4CXX, a c++-port of the java logging framework log4j.

- Make yourself familiar with Log4CXX (http://logging.apache.org/log4cxx/). Download the code of the library from the course website and build it with the autotools. *Remark:*
  - Additionally, you need the libraries apr1-dev and apr-util1-dev installed on your system.
- Think about a reasonable stucture for the loggers and use log-instructions instead of std::cout in your code

  For the configuration use a file in the property format.

Task 3 „Collision of two bodies “

- Implement a particle generator with which we can initialize the simulation. The generator should create a cuboid, where the molecules are arranged in a 3d rectangular grid.

  The cuboid is being described by the following parameters:
  - The coordinate of the lower left front-side corner.
  - Number of particles per dimension: $N_1 \times N_2 \times N_3$.
  - Distance h of the particles (mesh width of the grid)
  - Mass m of the particles
  - Initial velocity v of the particles (3 Components).
  - The mean-value of the velocity of the Brownian Motion. For now it is sufficient if you hard-code it in the code.

  The movement of the molecules should be superposed by Brownian Motion. Use the code from the website to add the thermal friction!

  Specify the parameters via the command line, or read them from a file. It has to be possible to add several cuboids! Moreover, it should be still possible to use a file for input.
- The interaction between two molecules i and j can be described with the help of the Lennard-Jones potential:

$$U(x_i, x_j) = 4 \cdot \epsilon \left(\left(\frac{\sigma}{||x_i - x_j||_2} \right)^{12} - \left(\frac{\sigma}{||x_i - x_j||_2} \right)^{6} \right)$$

.

The resulting force can be calculated as

$$F_{ij} = \frac{24 \cdot \epsilon}{(||x_i - x_j||_2)^2} \cdot \left(\left(\frac{\sigma}{||x_i - x_j||_2} \right)^{6} - 2 \cdot \left(\frac{\sigma}{||x_i - x_j||_2} \right)^{12} \right) \cdot (x_j - x_i)$$

.

Implement a new method for force calculation, which computes the Lennard-Jones force between two molecules! For the moment it’s ok if you hard-code the Lennard-Jones parameters ϵ und σ.