Project: Refactor Legacy ClassicMC-MoRiBs Software with Modern C++ Techniques

I. Overview

A. The legacy software

In 2016, I wrote a molecular Monte Carlo(MC) Simulation Software, named ClassicMC-MoRiBs, for a graduate school course project. The old software can be found at: https://github.com/Lechengwang/ClassicMC-MoRiBs

While the legacy project was called ‘C++’ and was compiled using g++, it was majorly using the plain C language features, except very little heap memory allocation. No classes, no smart pointers, no modern C++ features used.

The software is simulating the random walk of hydrogen molecules at low temperature inside a molecular cluster (cluster contains several molecules). The random walk is determined by the interaction between the molecules. By studying the random walk we can get the total interact energy for molecular cluster, and thus predict its boiling point, stability, and so on. Since we are majorly focused on refactoring using Modern C++ techniques, we are not going too deep in the physics theory.

If you are really interested in the physics background, see the uploaded PhysicsInsight.pdf. This is a presentation I presented for the software, physics background, and result, in 2016.

B. Current project

Currently project is refactoring the legacy software by the modern C++ techniques learned in Udacity C++ Nanodegree program, and is focused on the following areas:

- Refactoring loops/functions to make simulation more efficiency
- Applying OOD programming and encapsulating related variables and functions into classes
- Exploiting modern memory management techniques such as smart pointers

Details of the rubric points met is summarized in Section IV.

II. Build, run, and result/output

A. The instruction of compiling is done in the makefile. To build, simply follow:

step1: make clean. This will clean the existing .o files and the executables

step2: make. This will compile the mch2 executable

step3: directly run mch2.

B. The sprng, which is a more efficient random number generator library, is used. It has already compiled, handled in makefile, and compatible in most platforms, so no further update needed. In some rare case it is not working, please let me know and I will try to use std::rand.

C. All input files needed is provided in the root folder:

qmc.input //For simulation settings. See qmc.input for details

parah2.pot //Molecular interaction files

h2matrix.pos //Initial structure for Hydrogen matrix.

D. The output has 2 part:

1. The screen output. See OutputExample_Screen file for details.

a. The constructor/destructor calls are explicitly printed out.

b. The dumping of the qmc.input and summary of initial positions for verification purpose

c. The simulation acceptance ratio.

2. The energy output. See OutputExample_Energy file for details. During execution, you can find it in ./result/h2_sum.eng.txt

There is another ./result/h2.eng.txt. This file is for debug purpose only while it is kept there.

3. Verifying the correctness by output:

a. In Screen output, verify the acceptance ratio is between 0.3-0.5 if you are not changing anything in qmc.input.

b. In the Energy output, verify the energy is around -207. A short explanation is:

- according to h2matrix.pos, we have 4 hydrogen molecules here, thus are totally 6 pairwise interactions. (4*3/2=6).

- according to the interaction file parah2.pot, the minimal (most stable) interaction is -34.85. Thus -34.85*6=-209
- -207 is slightly higher than -209, since we allow random walk of the molecules, thus they are not exactly located at lowest energy positions – they are slightly moving around that position.

III. File structure

A. There are 5 classes defined:

1. MCSettings: MCSettings.h and MCSettings.cpp. This is the settings of the Monte Carlo simulation, which are initialized before the simulation.
2. RNDGenerator: RNDGenerator.h and RNDGenerator.cpp. This is the wrapper class for 'sprng' random number generator. The initialization of random stream, such as seed, is done in the constructor.
3. Potential: Potential.h and Potential.cpp: The 'molecular interaction energy' is formally called potential in physics. Its major function is compute pairwise molecular interaction by spline.
4. MCEstim: MCEstim.h and MCEstim.cpp. This is the energy estimator used in MonteCarlo simulation. By providing current configuration of molecular cluster, it can compute the total interaction by adding up pairwise energy, provided by Potential.SPot1D() method.
5. MCMover: MCMover.h and MCMover.cpp. This is the Monte Carlo mover simulating the random walk of molecules. It will compute if a random walk is acceptable, by checking the energy difference before and after the random walk. Also the energy is computed by the potential provide by Potential class.

B. There are several utility header files/cpp files. All named with prefix mc_, except mc_main.cpp which contains the main() method.

1. mc_config.h: including some configurations that are constant. (In contrast MCSettings class is handling the changeable settings)
2. mc_input.h and mc_input.cpp: including the static utility methods for input/output.
3. mc_utils.h and mc_utils.cpp: including other static utility methods, such as the spline function.

C. mc_main.cpp: The main method.

IV. Rubric Points

A. Loops, Functions, I/O

1. The project demonstrates an understanding of C++ functions and control structures.

Addressed in mostly all files. For example please see main() in main.cpp, line 57-85

2. The project reads data from a file and process the data, or the program writes data to a file.

Addressed as mentioned in subsection II.C and II.D. For code please see mc_input.cc, line 33-103 for file read, or main.cpp, line 137-158 for file write.

B. Object Oriented Programming

1. The project uses Object Oriented Programming techniques.

Addressed. See the five classes in Subsection III.A. For code example see Potential.h.

2. Classes use appropriate access specifiers for class members.

Addressed. See the five classes in Subsection III.A. For code example see Potential.h.

3. Class constructors utilize member initialization lists.

Addressed. For code example see MCMover.cpp, line 14-17; MCEstim.cpp, line 16. Note constructor of MCSettings, RNDGenerator, Potential does not take any parameters, since they are majorly initializing their own data structures.

4. Classes encapsulate behavior.

Addressed. See Potential.h and Potential.cpp for example. User are not aware of how the potential is initialized(Line 15-51, Potential.cpp) and computed (Line 60-76, Potential.cpp) and what private members are used (Line 18-27, Potential.h).

Also see MCSettings.h for example. All settings (Line 59-71) are accessible by getter only (Line 40-49).

C. Memory Management

1. The project makes use of references in function declarations.

Addressed. For example1, see Line 44 in mc_input.h and Line 29 in mc_input.cpp. For example2, see Line 30, 72, 88 in mc_main.cpp.

2. The project uses scope / Resource Acquisition Is Initialization (RAII) where appropriate.

Addressed. See Potential.cpp, Line 22-24 for allocation, Line 28 for initialization (using the utility function init_spline), and line 55-57 for destructor. This is also verified in Screen output. See line 17 and line 42 in OutputExample_Screen.

3. The project uses smart pointers instead of raw pointers.

Addressed. See Line 40, 43 and 45 in main.cpp for example, and Line 50 in main.cpp and line 12 in MCMover.cpp for how shared pointers are passed as arguments.