Programming Skills Coursework

Tim Beattie, David Canning, Marton Feigl, Max Sorantin

November 5, 2014

Contents

Here is an example of what the C++ source would look like using the listing:

```
#include<iostream>

int main (void)
{
   std::cout << "Hello, world!" << endl;

   return 0;
}
```

Writing equations:

$$E_{tot} = mc^2/\sqrt{1 - v^2/c^2} \tag{1}$$

We can then refer back to equation 1 like so.

1 Introduction

2 Planning

At the first meeting it was decided to set up version control, for details see sec.6, and to create a Tasklist. The latter was structured along the lines described in the "starting your project" section of the courserwork description and everyone added specific tasks he could think of to the variouse sections and mark them as essential, i.e. needed for a basic implementation of the project, or polish. After this Tasklist was established we agreed on a coding standard, build system and test framework, see sec.(7) and sec.(8), which was setup subsequentely. It was than decided on the overall programm design/layout, see sec.(4), which led to the aim of writing the needed code/-classes and their Unit Tests. With the essential part of the Tasklist therefore implemented the focus was planned to be shifted to the "polish" tasks starting with profiling and optimisation (see sec.(10).

However, in the real execution this was followed only up to the actual coding of the unit tests as profiling and optimisation was performed before carefull unit tests were in place.

3 Assignment of tasks to group members

4 Design

5 Programming Language

6 Revision Control

7 Build Tools

8 Testing

8.1 Unit testing

The project's unit testing was done with UnitTest++, available at

http://unittest-cpp.sourceforge.net

Quote from this page:

> *"UnitTest++ is a lightweight unit testing framework for C++. It was designed to do test-driven development on a wide variety of platforms. Simplicity, portability, speed, and small footprint are all very important aspects of UnitTest++."*

UnitTest++ was chosen because it was freely available, cross-platform, simple to set up and use, and equipped with a small but effective set of unit testing features.

8.1.1 Implementation of the unittest framework

UnitTest++ was downloaded as a .zip file and added to the project's Git repository in the "downloads" directory. UnitTest++ was then integrated into the project's build system. Running "make test" in the project's top-level directory will unzip the downloaded file to the directory "UnitTest++", call UnitTest++'s makefile to build the UnitTest++ static library, run its self-tests and build plus run the project's unit tests which link in that library. To ensure that the binary code being unit tested was the same code included in the popsim program, the project's build system was configured to build all C++ classes into a static library which was then linked into both the popsim program and the unit test program. This avoided problems which can occur when compiler or linker settings differ between multiple builds of the same source code. The project's unit test program was implemented in a small C++ implementation file which provided main() and included test headers for the project's C++ classes. Each class's test header contained one or more UnitTest++ "TEST" macros which implemented unit tests for that class. The main() function simply contained a call to UnitTest::RunAllTests(). All test code was placed in the "test" directory.

8.1.2 Unit tests for specific classes

The project was begun with the firm intention of writing each class's unit tests during the development of that class, ensuring that every feature of

every class would be robust and dependable before being used by the popsim program. However, as the deadline approached, pressure was felt and several key classes were written without tests, with the intention to write tests later. Due to this short comming, the team then lost considerable time on bug fixing, see sec.(9).

9 Debugging

10 Performance Tests and Analysis

11 Conclusions

Even thinking of unittests makes the design more split up and testable...

12 Further Work