統計物理・量子物理のためのプログラム開発技術

Techniques in Program Development for Statistical and Quantum Physics

藤堂眞治 (東京大学大学院理学系研究科) / Synge Todo (Department of Physics, University of Tokyo)

コンピュータは長く煩雑な作業を人間にかわって何桁も速く行ってくれる道具であり、プログラムはその道具の設計図である。プログラムは書いた通りに動く。しかし、思った通りにはなかなか動いてくれない。そもそもプログラムは人間が楽をするために作るもので、プログラム開発に長い時間がかかったり、苦痛を感じることがあってはならない。動作するきれいなコードを楽に開発するにはどうすればよいのだろうか？プログラム開発に関する教科書を読むと、プログラムの内部構造やデータ構造に関する仕様をあらかじめきちんと定め、ドキュメントとして残したり、コードの各所に適切なコメントを付けたりすることが大切であると強調されている。しかしながら、コードを書き始める前に細部の設計を済ませたり、完全なドキュメントを維持するのは、たとえプログラム開発に関する専門教育を受けた人であっても難しい。

この問題の解決にむけたソフトウェア開発手法として、「テスト駆動開発」がある。テスト駆動とは、単にテストコードをたくさんつけることではない。テスト駆動開発においては、まずテストを書くことで、プログラムの機能や仕様をテストコードとして明確に表現する。次に、最低限のコードを書き、そのテストをクリアする。そして、すべてのテストがパスすることを確認しながら、コードの重複を取り除いていく。この手順を繰り返すことで、速くて正確に動作し、きれいで再利用可能なコードを作ることができる。また、プログラムの仕様や使い方などのドキュメントも、実際にコードを書き始める前にあらかじめ出来上がっていることになる。

本講義では、このテスト駆動開発手法を実例を交えながら解説する。テスト駆動開発により、不安をかかえこまずに、欠陥の少ないプログラムを開発することが可能になる。この開発手法は、小さなツールから非常に大規模なソフトウェアまで、あるいは、個人開発から大きなグループでの開発まで幅広く活用することができる。実際のテスト駆動開発では、何度も繰り返しプログラムをコンパイルし、大量のテストを実行すことになる。本講義では、ビルドシステム、テストフレームワーク、継続的インテグレーション、バージョン管理システムなど、テスト駆動開発をサポートするツールや環境についても紹介する。

A computer is a tool that performs long and tedious tasks for humans many orders of magnitude faster, and a program is a blueprint for that tool. Programs work as we wrote, but they do not always work as expected. Writing a program should not take long or not be painful as we use them to make life easier. How can we effortlessly develop working clean code? Textbooks on program development emphasize the importance of having well-defined specifications for the program's internal structure and data structure in advance, documenting them, and adding appropriate comments throughout the code. However, it is not easy, even for people with specialized training in program development, to design the details in advance and maintain documentation.

Test-driven development (TDD) is a development method that aims to solve these problems. TDD means more than just writing lots of test code. In TDD, we write tests first to define program functions and specifications in the form of code. Next, we write a minimum amount of code to pass the tests. Then, while ensuring that all tests pass, we refactor the code by removing duplications. By repeating this procedure, we can create code that is fast, works correctly, and is clean and reusable. In addition, documentation of the program specifications and usage will have been available before we start writing the code.

This lecture explains TDD with actual examples. TDD makes it possible to develop programs with fewer defects without being anxious. It can be used in various situations, from small tools to huge software, from individual to large-group development. In actual TDD, we compile programs over and over again and run a large number of tests. This lecture will also introduce tools and environments that support TDD, such as build systems, test frameworks, continuous integration, and version control systems.