Δομές Δεδομένων και Αλγόριθμοι - Παραρτήματα

Τ.Ε.Ι. Ηπείρου, Τμήμα Μηχανικών Πληροφορικής Τ.Ε.
Χρήστος Γκόγκος - Αναπληρωτής Καθηγητής

1 Test Driven Development

Τα τελευταία χρόνια η οδηγούμενη από ελέγχους ανάπτυξη (Test Driven Development) έχει αναγνωριστεί ως μια αποδοτική τεχνική ανάπτυξης εφαρμογών. Αν και το θέμα είναι αρκετά σύνθετο, η βασική ιδέα είναι ότι ο προγραμματιστής πρώτα γράφει κώδικα που ελέγχει αν η ζητούμενη λειτουργικότητα ικανοποιείται και στη συνέχεια προσθέτει τον κώδικα που θα υλοποιεί αυτή τη λειτουργικότητα [1]. Ανά πάσα στιγμή υπάρχει ένα σύνολο από συσσωρευμένους ελέγχους οι οποίοι για κάθε αλλαγή που γίνεται στον κώδικα είναι σε θέση να εκτελεστούν άμεσα και να δώσουν εμπιστοσύνη μέχρι ένα βαθμό στο ότι το υπό κατασκευή ή υπό τροποποίηση λογισμικό λειτουργεί ορθά. Γλώσσες όπως η Java και η Python διαθέτουν εύχρηστες βιβλιοθήκες που υποστηρίζουν την ανάπτυξη TDD (junit και pytest αντίστοιχα). Στην περίπτωση της C++ επίσης υπάρχουν διάφορες βιβλιοθήκες που μπορούν να υποστηρίξουν την ανάπτυξη TDD. Στα πλαίσια του εργαστηρίου θα χρησιμοποιηθεί η βιβλιοθήκη Catch (https://github.com/philsquared/Catch) για το σκοπό της επίδειξης του TDD.

Στο παράδειγμα που ακολουθεί παρουσιάζεται η υλοποίηση της συνάρτησης παραγοντικό. Το παραγοντικό συμβολίζεται με το θαυμαστικό (!), ορίζεται μόνο για τους μη αρνητικούς ακεραίους αριθμούς και είναι το γινόμενο όλων των θετικών ακεραίων που είναι μικρότεροι ή ίσοι του αριθμού για τον οποίο ζητείται το παραγοντικό. Το δε παραγοντικό του μηδενός είναι εξ ορισμού η μονάδα. Η πρώτη υλοποίηση είναι λανθασμένη καθώς δεν υπολογίζει σωστά το παραγοντικό του μηδενός αποδίδοντάς του την τιμή μηδέν.

```cpp
#define CATCH_CONFIG_MAIN // This tells Catch to provide a main() - only do this
                          // in one cpp file
#include "catch.hpp"

unsigned int Factorial(unsigned int number) {
  return number <= 1 ? number : Factorial(number - 1) * number;
}

TEST_CASE("Factorials are computed", "[factorial]") {
  REQUIRE(Factorial(0) == 1);
  REQUIRE(Factorial(1) == 1);
  REQUIRE(Factorial(2) == 2);
  REQUIRE(Factorial(3) == 6);
  REQUIRE(Factorial(10) == 3628800);
}
```

Κώδικας 1: Πρώτη έκδοση της συνάρτησης παραγοντικού και έλεγχοι (tdd1.cpp)

Η μεταγλώττιση και η εκτέλεση του κώδικα γίνεται ως εξής:

```
g++ tdd1.cpp -o tdd1
./tdd1
```

Τα δε αποτελέσματα της εκτέλεσης είναι τα ακόλουθα:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
tdd1 is a Catch v1.10.0 host application.
Run with -? for options
```

```
-------------------------------------------------------------------------------
Factorials are computed
-------------------------------------------------------------------------------
tdd1.cpp:9
.....................................................................

tdd1.cpp:10: FAILED:
  REQUIRE( Factorial(0) == 1 )
with expansion:
  0 == 1

===============================================================================
test cases: 1 | 1 failed
assertions: 1 | 1 failed
```

Η δεύτερη υλοποίηση είναι σωστή. Τα μηνύματα που εμφανίζονται σε κάθε περίπτωση υποδεικνύουν το σημείο στο οποίο βρίσκεται το πρόβλημα και ότι πλέον αυτό λύθηκε.

```
#define CATCH_CONFIG_MAIN // This tells Catch to provide a main() - only do this
                          // in one cpp file
#include "catch.hpp"

unsigned int Factorial(unsigned int number) {
  return number > 1 ? Factorial(number - 1) * number : 1;
}

TEST_CASE("Factorials are computed", "[factorial]") {
  REQUIRE(Factorial(0) == 1);
  REQUIRE(Factorial(1) == 1);
  REQUIRE(Factorial(2) == 2);
  REQUIRE(Factorial(3) == 6);
  REQUIRE(Factorial(10) == 3628800);
}
```

Κώδικας 2: Δεύτερη έκδοση της συνάρτησης παραγοντικού και έλεγχοι (tdd2.cpp)

```
===============================================================================
All tests passed (5 assertions in 1 test case)
```

Αναφορές

[1] http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd