Razvoj softvera - Čas 6

Primeri sa predavanja iz predmeta *Razvoj softvera* na Matematičkom fakultetu Univerziteta u Beogradu u školskoj 2023/24. godini.

Prof. dr Saša Malkov

U svakom primeru ćemo navoditi samo one fajlove koji se menjaju. Neizmenjene fajlove nećemo da ponavljamo. Uvek se prvo menja `Vektor.test.cpp`, pa onda `Vektor.h`, pa na kraju `Vektor.cpp`.

Primeri se grade skriptovima `run.cmd` (Windows) i `run.sh` (Linux).

run.cmd

```
@echo off

call cl -c Vektor.cpp /EHsc /nologo
if errorlevel 1 goto :end

call cl -c Vektor.test.cpp /EHsc /nologo
if errorlevel 1 goto :end

call cl Vektor*.obj main.test.cpp /Fe:test.exe /EHsc /nologo
if errorlevel 1 goto :end

echo.
echo --- RUN ---
call test.exe

del *.obj > nul

:end
exit /b
```

run.sh

```
g++ Vektor.cpp Vektor.test.cpp main.test.cpp -o test
if [ $? -eq 0 ]; then
    ./test
fi
```

Korak v01

main.test.cpp

```
#define CATCH_CONFIG_MAIN
#include "../catch.hpp"
```

Vektor.cpp

```
#include "Vektor.h"
```

Vektor.h

```
#ifndef VEKTOR_H_
#define VEKTOR_H_

class Vektor;

#endif // #ifndef VEKTOR_H_
```

Vektor.test.cpp

```
#include "../catch.hpp"
#include "Vektor.h"
```

Korak v02

Vektor.h

```
#ifndef VEKTOR_H_
#define VEKTOR_H_

class Vektor
{
public:
   Vektor( double x, double y, double z )
      : X_(x), Y_(y), Z_(z)
      {}

   double getX() const
      { return X_; }

   double getY() const
      { return Y_; }

   double getZ() const
      { return Z_; }
```

```
private:
   double X_, Y_, Z_;
};

#endif // #ifndef VEKTOR_H_
```

Vektor.test.cpp

```
#include "../catch.hpp"
#include "Vektor.h"

TEST_CASE( "Konstruktor", "[Vektor]" ) {
   Vektor v(1,2,3);
   REQUIRE( v.getX() == 1 );
   REQUIRE( v.getY() == 2 );
   REQUIRE( v.getZ() == 3 );
}
```

Korak v03

Vektor.test.cpp

```
#include "../catch.hpp"
#include "Vektor.h"

TEST_CASE( "Konstruktor", "[Vektor]" ) {
   Vektor v(1,2,3);
   REQUIRE( v.getX() == 1 );
   REQUIRE( v.getY() == 2 );
   REQUIRE( v.getZ() == 3 );

   //  namerno, da vidimo kako izgleda kada se pogresi
   REQUIRE( v.getX() == 3 );
}
```

Korak v04

Vektor.h

```
#ifndef VEKTOR_H_
#define VEKTOR_H_

class Vektor
```

```
{
public:
   Vektor()
      : X_(0), Y_(0), Z_(0)
      {}

   Vektor( double x, double y, double z )
      : X_(x), Y_(y), Z_(z)
      {}

   double getX() const
      { return X_; }

   double getY() const
      { return Y_; }

   double getZ() const
      { return Z_; }

private:
   double X_, Y_, Z_;
};

#endif // #ifndef VEKTOR_H_
```

Vektor.test.cpp

```
#include "../catch.hpp"
#include "Vektor.h"

TEST_CASE( "Konstruktori", "[Vektor]" ) {
    SECTION( "Vektor()" ){
        Vektor v;
        // Poredjenje realnih brojeva sa nulom je obicno u redu
        CHECK( v.getX() == 0 );
        CHECK( v.getY() == 0 );
        CHECK( v.getZ() == 0 );
        //  Ali u opstem slucaju bi bilo bolje ovako
        Approx doubleTest = Approx(0).epsilon(0.0000001);
        CHECK( v.getX() == doubleTest );
        CHECK( v.getY() == doubleTest );
        CHECK( v.getZ() == doubleTest );
    }
    SECTION( "Vektor(double,double,double)" ){
        Vektor v(1,2,3);
        CHECK( v.getX() == 1 );
        CHECK( v.getY() == 2 );
        CHECK( v.getZ() == 3 );
    }
}
```

Korak v05

Vektor.h

```
#ifndef VEKTOR_H_
#define VEKTOR_H_

class Vektor
{
public:
   Vektor()
      : X_(0), Y_(0), Z_(0)
      {}

   Vektor( double x, double y, double z )
      : X_(x), Y_(y), Z_(z)
      {}

   double getX() const
      { return X_; }

   double getY() const
      { return Y_; }

   double getZ() const
      { return Z_; }

   bool operator==( const Vektor& v ) const
   {
      return getX() == v.getX()
         && getY() == v.getY()
         && getZ() == v.getZ();
   }

   bool operator!=( const Vektor& v ) const
      { return !(*this == v); }

private:
   double X_, Y_, Z_;
};

#endif // #ifndef VEKTOR_H_
```

Vektor.test.cpp

```
#include "../catch.hpp"
#include "Vektor.h"

TEST_CASE( "Konstruktori", "[Vektor]" ) {
```

```
...
}

TEST_CASE( "Operatori", "[Vektor]" ){
   Vektor v(1,2,3);
    Vektor v1(2,2,3);
    Vektor v2(1,3,3);
    Vektor v3(1,2,2);

   SECTION( "operator==" ) {
      CHECK( v == v );
      CHECK_FALSE( v == v1 );
      CHECK_FALSE( v == v2 );
      CHECK_FALSE( v == v3 );
   }

   SECTION( "operator!=" ) {
      CHECK_FALSE( v != v );
      CHECK( v != v1 );
      CHECK( v != v2 );
      CHECK( v != v3 );
   }

}
```

Korak v06

Vektor.cpp

```
#include "Vektor.h"

using namespace std;

ostream& operator<<( ostream& ostr, const Vektor& v )
{
   return ostr << '(' << v.getX() << "," << v.getY() << "," << v.getZ() << ")";
}

istream& operator>>( istream& istr, Vektor& v )
{
   double x,y,z;
   char c1,c2,c3,c4;
   istr >> c1 >> x >> c2 >> y >> c3 >> z >> c4;
   if( istr && c1 == '(' && c2 == ',' && c3 == ',' && c4 == ')' ){
      v.X_ = x;
      v.Y_ = y;
      v.Z_ = z;
   }
    else
      istr.setstate( ios::failbit );
```

```
   return istr;
}
```

Vektor.h

```
#ifndef VEKTOR_H_
#define VEKTOR_H_

#include <iostream>

class Vektor
{
public:
   Vektor()
      : X_(0), Y_(0), Z_(0)
      {}

   Vektor( double x, double y, double z )
      : X_(x), Y_(y), Z_(z)
      {}

   double getX() const
      { return X_; }

   double getY() const
      { return Y_; }

   double getZ() const
      { return Z_; }

   bool operator==( const Vektor& v ) const
   {
      return getX() == v.getX()
         && getY() == v.getY()
         && getZ() == v.getZ();
   }

   bool operator!=( const Vektor& v ) const
      { return !(*this == v); }

private:
   double X_, Y_, Z_;

   friend std::istream& operator>>( std::istream& ostr, Vektor& v );
};

std::ostream& operator<<( std::ostream& ostr, const Vektor& v );
std::istream& operator>>( std::istream& istr, Vektor& v );

#endif // #ifndef VEKTOR_H_
```

Vektor.test.cpp

```
#include "../catch.hpp"
#include "Vektor.h"

#include <sstream>
using namespace std;

TEST_CASE( "Konstruktori", "[Vektor]" ) {
...
}

TEST_CASE( "Operatori", "[Vektor]" ){
   Vektor v(1,2,3);
    Vektor v1(2,2,3);
    Vektor v2(1,3,3);
    Vektor v3(1,2,2);
    Vektor v4(1.1,2.2,3.3);

   SECTION( "operator==" ) {
      CHECK( v == v );
      CHECK_FALSE( v == v1 );
      CHECK_FALSE( v == v2 );
      CHECK_FALSE( v == v3 );
   }

   SECTION( "operator!=" ) {
      CHECK_FALSE( v != v );
      CHECK( v != v1 );
      CHECK( v != v2 );
      CHECK( v != v3 );
   }

   SECTION( "operator<<" ) {
      ostringstream ostr;
      ostr << v4;
      CHECK( ostr );
        CHECK( ostr.str() == "(1.1,2.2,3.3)" );
   }

   SECTION( "operator<<" ) {
      istringstream istr( "(1.1,2.2,3.3)" );
      Vektor v5;
      istr >> v5;

      CHECK( v5 == v4 );
      CHECK( istr );

      char c;
      istr >> c;
      CHECK( !istr );
   }
```

```
}
```

Korak v07

Vektor.test.cpp

```
#include "../catch.hpp"
#include "Vektor.h"

#include <sstream>
using namespace std;

TEST_CASE( "Konstruktori", "[Vektor]" ) {
...
}

TEST_CASE( "Operatori", "[Vektor]" ){
...
}

TEST_CASE( "Razno 2", "[test][vektor]" ) {
   Vektor v(1,2,3);
   INFO("Na ovom primeru testa mogu da se isprobaju tagovi...");
   INFO("...pokrenite ./test -?");
   INFO("Naredni makro ispisuje vektor v:");
   CAPTURE( v );
    INFO("Sve ovo se ispisuje samo u slucaju greske, pa je namerno pravimo...");
   REQUIRE_FALSE( v ==  v );
}
```

Korak v08

Vektor.test.cpp

```
#include "../catch.hpp"
#include "Vektor.h"

#include <sstream>
using namespace std;

TEST_CASE( "Konstruktori", "[Vektor]" ) {
...
}

TEST_CASE( "Operatori", "[Vektor]" ){
```

```
...
}

TEST_CASE( "Razno 2", "[test][vektor]" ) {
   Vektor v(1,2,3);
   INFO("Na ovom primeru testa mogu da se isprobaju tagovi...");
   INFO("...pokrenite ./test -?");
   INFO("Naredni makro ispisuje vektor v:");
   CAPTURE( v );
    INFO("Sve ovo se ispisuje samo u slucaju greske, pa je namerno pravimo...");
   REQUIRE_FALSE( v ==  v );
}

TEST_CASE( "Razno", "[test]" ) {
   // ne postoji poseban makro za proveru jednakosti
   // ali postoji posebna konstrukcija za poredjenje realnih brojeva

    // ovaj test ne prolazi
    CHECK( (10.0/3) == Approx(3.33) );

    // ovaj test prolazi
   Approx floatTest = Approx(3.33).epsilon(0.01);
   CHECK( (10.0/3) == floatTest );  // dopusten procenat odstupanja

   // REQUIRE_FAIL(...);
   // REQUIRE_NOTHROW(...);
   // REQUIRE_THROWS(...);
}
```