Unit testing MathVector

Prof. Y C Cheng
2009/10/21

國立台北科技大學 資訊工程系
National Taipei University of Technology
Dept of Comp Sci and Info Engr
Math Vector

• Compute the dot product of two vectors of the same dimension. For example,

\[(1, 0) \cdot (1, 1) = 1\]
\[(1, 1, 0) \cdot (0, 1, 1) = 1\]
\[(1,0)(1,1,0) \text{ is illegal}\]

Length of a vector: \[\|(1,1,0)\| = \sqrt{2}\]
Unit vector of \((1,1,0) = (1,1,0)/\sqrt{2}\]
Devise a plan – list the tasks

1. As a programmer, I can write the main program
3. v As a programmer, I can define a n-dimensional vector with given values for its components
4. v As a programmer, I can print a vector.
5. v As a programmer, I can compute the dot product of two vectors of the same dimension
7. v As a programmer, I can detect if two vectors can have a dot product
8(3). v As a programmer, I can define a n-dimensional vector with given values for its components with class.
9(4). v As a programmer, I can print a class-version vector.
10. As a programmer, I can compute the length of a vector
11. As a programmer, I can multiply a vector by a scalar.
12. As a programmer, I can compute the unit vector of a given vector.
Looking back – can we improve the program?

- We need other functions of the MathVector
- But it’s getting difficult to move around in the one and only file containing 180 lines of code
- Also, it is getting difficult to decipher the “output” of the tests we wrote in “main()”
Understand the problems

• Change the program structure while making sure that the program still works!
  – Move tests in main into unit tests with help from CppUnitLite
  – Separate MathVector from the rest of the program by making the change in small steps (refactoring)

• Then, we are going to add other member functions (e.g., destructors and copy constructor)
New structure with unit tests

main.cpp

MathVector.h
MathVector.cpp
MathVectorTest.cpp
main.cpp

The unit tests, organized by (Class Name, member function name)
What do we gain with the new structure?

- Identified places to add things rather than moving around in a single big file
- Make it easy to remember the actions:
  1. Write test first
  2. Write declaration of a member functions, etc.
  3. Write implementation of the declared member function, etc.
  4. If the test passes, you’re done (move on to the next member function); otherwise repeat 2 and 3.

Test driven!
Steps to add a new member function

• To add a new member function “length” to MathVector:
  – Adding a new test case in MathVectorTest.cpp:
    TEST( MathVector, length )
    {
      MathVector w2(2,1,1);
      DOUBLES_EQUAL (1.41421, w2.length(), 0.0001);
    }
  – Adding prototype declaration of “length” in MathVector.h
    double length();
  – Adding definition (implementation) of the new member function in
    MathVector.cpp
    double MathVector::length() { … }
  – Compile and run
  – Repeat if failure; else move on or add more tests!
Devise a plan – list the tasks

1. As a programmer, I can write the main program
3. As a programmer, I can define a n-dimensional vector with given values for its components

... 
10. As a programmer, I can compute the length of a vector
11. As a programmer, I can multiply a vector by a scalar.
12. As a programmer, I can compute the unit vector of a given vector.
13(10). As a programmer, I can test that length of a vector is computed correctly.
14(11). As a programmer, I can test the scalar product of a vector.
15(12). As a programmer, I can test that the unit vector is computed correctly for a given vector
13(10). As a programmer, I can test that length of a vector is computed correctly.

- Add library path of CppUnitLite
  - E.g., project -> project options -> Parameters -> add library or objects -> select CppUnitLight.a

- Normally, you wrote tests in a separate file but for now, let’s add all tests in “main.cpp”
  - Later, we will relocate the tests in a separate file
Task 13, continued.

• First, you make CppUnitLite available by adding the following line to main.cpp

```cpp
#include <cstdlib>
#include <iostream>
#include <cmath>
#include "C:\Dev-Cpp\CppUnitLite\TestHarness.h"

using namespace std;

class MathVector {
public:
    // constructor 2-d vector
    MathVector(int dim, double c1, double c2) :
        _dim(dim)
    {
```
Task 13, continued.

- Add the call to execute unit tests in main.cpp

```cpp
double length = w2.length();
cout << "length of w2 = " << length << endl;
MathVector *pv = w2.scalarMultiply(1/length);
printVector(*pv);
MathVector *puv = w2.unitVector();
printVector(*puv);
TestResult tr;
TestRegistry::runAllTests(tr);
system("PAUSE");
```
Task 13, add test

• In main.cpp, add

```cpp
TEST( MathVector, length )
{
    MathVector w2(2,1,1);
    DOUBLES_EQUAL (1.41421, w2.length(), 0.0001);
}
```

Defined in CPPUnitLite/Test.h:

Expected result by calling the “length()”

Actual: returned member function “length()”

Threshold for testing equality of two doubles
Task 13, continued.

- Declare “length()”: already done.
- Implement “length()”: already done.
- Compile and run.
14(11). As a programmer, I can test the scalar product of a vector.

```cpp
TEST( MathVector, scalarMultiply )
{
    MathVector w2(2,1,1);
    double length = w2.length();
    MathVector *pv = w2.scalarMultiply(1/length);
    printVector(*pv);
    DOUBLES_EQUAL (0.707107, pv->_component[0], 0.0001);
    DOUBLES_EQUAL (0.707107, pv->_component[1], 0.0001);
    delete pv;
}
```
15(12). As a programmer, I can test the unit vector for a given vector

```cpp
TEST( MathVector, unitVector )
{
    MathVector w2(2,1,1);
    MathVector *pv = w2.unitVector();

    DOUBLES_EQUAL (0.707107, pv->_component[0], 0.0001);
    DOUBLES_EQUAL (0.707107, pv->_component[1], 0.0001);

    delete pv;

    pv = MathVector(2,0,0).unitVector();
    DOUBLES_EQUAL (0.0, pv->_component[0], 0.0001);
    DOUBLES_EQUAL (0.0, pv->_component[1], 0.0001);
    delete pv;
}
```
Clean up main.cpp

• After make sure all tests passed, remove (or comment out) the lines in main.cpp corresponding to the tests
• Effectively, we have moved the tests out “main()”

```cpp
/*
 * double length = w2.length();
 * cout << "length of w2 = " << length << endl;
 *
 * MathVector *pv = w2.scalarMultiply(1/length);
 * printVector(*pv);
 *
 * MathVector *pvu = w2.unitVector();
 * printVector(*pvu);
 */
```
Compile and run...

There were no test failures

請按任意鍵繼續 . . .
Homework #4

• Repeat the steps until all tests in the main.cpp (in CVector, version 4 downloaded from my homepage) is moved out of “main()” into a number of TEST as we did in this set of slides.
• Remember to write up the tasks in todo.txt first
Separating MathVector out of main.cpp

• We assume that you have created the required tests. All tests now pass.
• With the tests in place, we are now safe to move on
  – If you break anything, either compilation fails or (hopefully) the unit tests fail
• We want code for MathVector to be extracted from part of main.cpp into two files:
  – MathVector.h, called the header file, containing the declarations for class MathVector;
  – MathVector.cpp, called the header file, containing the declarations for class MathVector;
Separating MathVector, continued

• But let’s do it one step at a time!

  15(12). v As a programmer, I can test that the unit vector is computed correctly for a given vector

  ...

  30. Extract code for MathVector into MathVector.h
  31. Extract implementation code for MathVector into MathVector.cpp
  32. Extract tests into MathVectorTest.cpp
Expected structure after task 30

main.cpp

MathVector.h

MathVector.cpp

MathVectorTest.cpp

main.cpp
30. Extract code for MathVector into MathVector.h

- Move all code for MathVector code and c functions (dot products, etc) into MathVector.h
  - Safe move: copy -> comment -> compile -> run -> remove from main.cpp
- Include MathVector.h in main.cpp
MathVector.h

```cpp
#ifndef MATH_VECTOR_H
#define MATH_VECTOR_H

#include <cstdlib>
#include <iostream>

using namespace std;

class MathVector {
public:
    // member functions and data members
};
#endif
```

Preventing multiple declaration of MathVector due to multiple inclusion; thereby avoiding compilation errors.
main.cpp, after separating MathVector:

```cpp
#include <cstdlib>
#include <iostream>
#include <cmath>
#include "C:\Dev-Cpp\CppUnitLite\TestHarness.h"
#include "MathVector.h"

using namespace std;

int main(int argc, char *argv[]) {
    TestResult tr;
    TestRegistry::runAllTests(tr);
    system("PAUSE");
    return EXIT_SUCCESS;
}

TEST( MathVector, length ) {
    MathVector w2(2,1,1);
    DOUBLES_EQUAL(1.41421, w2.length(), 0.0001);
    MathVector w5(3,1,1,1);
```
31. Extract implementation code for MathVector into MathVector.cpp

- Create MathVector.cpp
- Move Implementation code (functions) into MathVector.cpp
  - Remember the Safe move: copy -> comment -> compile -> run -> remove
Running into trouble

TEST( MathVector, dotProduct )
{
    MathVector w1(2,1,1), w2(2,1,0);
    DOUBLES_EQUAL (1.0, dotProduct(w1,w2),
    MathVector w3(3,1,1,0);
    double r;
    r = dotProduct(w1, w2);
}

In member function `virtual void dotProductMathVectorTest::run(TestResult&)':
`dotProduct' undeclared (first use this function)

(Each undeclared identifier is reported only once for each function it appears in.)

Build Error [main.o] Error 1
Fixing “undeclared dotProduct”

- Add functional prototype of dotProduct (and other functions) in MathVector.h

```cpp
void printVector(double v[], int dim);
void printVector(MathVector v);
double dotProduct(MathVector u, MathVector v);
bool dotProduct(MathVector u, MathVector v, double & r);

/*
void printVector(double v[], int dim)
{
    cout << "(";
    for (int i=0; i< dim-1; ++i)
        cout << v[i] << ",";
    cout << "\n";
}
*/
```
More (minor) trouble
Structure after task 31

main.cpp

MathVecor.h

MathVecor.cpp

MathVecorTest.cpp

main.cpp
32. Extract tests into MathVectorTest.cpp

```
#include "C:\Dev-Cpp\CppUnitLite\TestHarness.h"
#include "MathVector.h"

TEST( MathVector, length )
{
    MathVector w2(2,1,1);
    DOUBLES_EQUAL (1.41421, w2.length(), 0.0001);
    MathVector w5(3,1,1,1);
    DOUBLES_EQUAL (1.7320, w5.length(), 0.0001);
}

TEST( MathVector, scalarMultiply )
{
    MathVector w2(2,1,1);
    double length = w2.length();
```
Main.cpp, after step 32

```cpp
#include <cstdlib>
#include "C:\Dev-Cpp\CppUnitLite\TestHarness.h"
#include "MathVector.h"

using namespace std;

int main(int argc, char *argv[])
{
    TestResult tr;
    TestRegistry::runAllTests(tr);
    system("PAUSE");
    return EXIT_SUCCESS;
}
```
Compile and run...
#include "MathVecotr.h"

Through the CppUnitLite library

#include “MathVecotr.h”
Now, complete Homework #4

• Change the one-file “main.cpp” into the four-file structure, “MathVector.h”, “MathVector.cpp”, “MathVectorTest.cpp”, and “main.cpp”

• Remember to write up the tasks in todo.txt first