Operator overload and Design for Testability

Prof. Y C Cheng
2012/10/24

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
Overloading operator []

• How to handle out of bound index?
  – Cannot use the return value as error code because it is reserved to return the reference to the component
  – Cannot use argument to return error code because operator[] (unsigned int)
• Use exception to convey error and enable error handling
try {
    callee.method();
} catch (exception) {
    // handle exception
}
Unit test: Silent unless broken

• Proper use of unit tests should eliminate verbose screen dump.
• Strive to keep unit tests so
Looking back...

• Vector operations: +, -, =, etc. ?
• Replace printVector with “<<“ so that we can write

  double a[] = {1,2};
  Vector v(a,2);
  cout << “The vector is” << v << endl;

• We will look at overloading (functions or operators with the same name) operator << and design for testability
Devise a plan – list the tasks

33. Add two vector with + (C function)
34. vector u - vector v- (member function)
35. overload the assignment operator = as a member function
36. output Vector with <<
37. add mf to convert Vector to a string (DFT)
38(36). Rewrite << so that it is tested.
33. Add two vector with + (C function) (1/2)

The unit test:
TEST(Vector, c_operator_plus)
{
    double a[] = {2,1};
    double b[] = {1,2};

    Vector v1(a,2);
    Vector v2(b,2);

    DOUBLES_EQUAL(3, (v1+v2).component(1),0.0001);
    DOUBLES_EQUAL(3, (v1+v2).component(2),0.0001);
}
33. Add two vector with + (C function) (2/2)

The declaration and implementation:

```cpp
const Vector operator +(const Vector & u, const Vector & v);

const Vector operator + (const Vector & u, const Vector & v)
{
    assert(u.getDim() == v.getDim());
    double a[u.getDim()];
    for (int i=1; i<=u.getDim(); ++i)
        a[i-1] = u.component(i) + v.component(i);
    return Vector(a, u.getDim());
}
```
34. vector u - vector v  
(member function)

The unit test:

TEST(Vector, member_function_operator_minus)
{
    double a[] = {2,1};
    double b[] = {1,2};

    Vector v1(a,2);
    Vector v2(b,2);

    DOUBLES_EQUAL(1, (v1-v2).component(1),0.0001);
    DOUBLES_EQUAL(-1, (v1-v2).component(2),0.0001);
}
34. vector u - vector v
(member function)

The implementation:

const Vector Vector::operator - (const Vector & u)
{
    double a[dim];
    for (int i=0; i < dim; ++i)
        a[i] = v[i] - u.v[i];

    return Vector(a,dim);
}
Which is better, a C function or a C++ member function?

- Due to encapsulation:
- More difficult for C function to access Vector data:
  - C function takes out the component values, do addition at the outside (stored in an C array), and create the required vector object at the last step.
- C++ mf has better access to elements,
  - Accessing components through private data member v
The Decision to encapsulate

• Cannot be all good, somewhere along the line it makes you pay!
• Thus, the decision to declare some members “private” is a deign decision.
36. output Vector with <<

The test:

TEST(Vector, insertion_operator)
{
    double a[] = {2,1};
    double b[] = {1,2,3};
    Vector v1(a,2);
    Vector v2(b,3);

    // not good enough?
    cout << v1 << endl;
    cout << v1 << " " << v2 << endl;
}

ostream cout; // in iostream.h
...
((((cout << v1) << " " )<< v2) << endl);
36. output Vector with << implementation:

```cpp
os << "(";
for (int i=1; i< vec.getDim(); ++i)
os << vec.component(i) << ",";
os << vec.component(vec.getDim()) << ")";

return os;
```
How do we test "<<"?

• Dumping to console seems to break our tests commitment to be “silent unless broken”:
  – now we have to look at the output!
• One solution to test “<<“ is to have Vector converted to a string.
• Then you can test the string is exactly as wished.
• Finally, you modify the overload “<<“ to use the string conversion.
37. add mf to convert Vector to a string

```
#include <sstream>

string Vector::toString() const
{
    ostringstream sos;
    sos << "(
    for (int i=1; i< this->getDim(); ++i)
        sos << this->component(i) <<"",
    sos << this->component(this->getDim()) <<""
    return sos.str();
}
```

Use:
cout << v.toString() << …;