COMP220 - Lab 7 & Hwk 6

Fall 2015

Abstract

For this lab you'll carry out Exercise 2 in chapter 6 of the text (pg 307). This exercise gives you practice doing basic class-based ADT design and exposes you to C++11 enumerated class types (*enum class*). Traditional enumerated types are discussed in chapter 1 section 6 (pg 24). The C++11 standard offers an improved enumerated type design so we'll go ahead with this updated, C++11 version. I recommend you checkout `http://www.cprogramming.com/c++11/c++11-nullptr-strongly-typed-enum-class.html` to get a feel for the how and why of the *enum class* for C++11.

1 The Problem

When the author says "export" he means provide the declaration and definition in an accessible way. For class methods, this means making public class methods. For Constants, this means declaring constant variables, for enum types, this means declaring the type. We will also use the C++11 *enum class* as opposed to strict *enum.*

For example, putting something like the following in a header will "export"

- the constant *example::PI_FOR_INDIANA*
- the enum class *example::Suit*
- and the class *example::myClass*

```
namespace example{

   // a constant double for "pi"
   const double PI_FOR_INDIANA = 3.2;

   // the suit enum
   enum class Suit { CLUBS, DIAMONDS, HEARTS, SPADES };

   class myClass{
      // class declaration code in here
   };
}
```

You can then refer to these definitions in the usual manner:

```
TEST(stuff,all){

  //constants value by name
  EXPECT_DOUBLE_EQ(3.2,example::PI_FOR_INDIANA);

  // the enumerated values by name
  EXPECT_EQ(0,example::Suit::CLUBS);
  EXPECT_EQ(1,example::Suit::DIAMONDS);
  EXPECT_EQ(2,example::Suit::HEARTS);
  EXPECT_EQ(3,example::Suit::SPADES);

  // Using the new class
  example::myClass a;
  example::myClass b;
  EXPECT_EQ(a,b);

}
```

It's important to note that constant and enum class definitions do not require anything additional code in the library cpp file. They're completely defined in the header. This is different than procedures and classes.

In addition to carrying out the requirements as stated in the book, you must:

- overload the == and ! = operator for the Card ADT
- write mutators/setters *setRank* and *setSuit*.

Some other things you might try (but do not have to for lab/hwk) include:

- Overload *operator*< for comparing cards. Suits are ranked in alphabetical order, i.e. the same order we declared them in the enum class above.
- Overload the operators << and >> for writing and reading cards to streams.

For lab, focus on the declarations, documentations, stubs, and tests. Submit these items as *lab7* using *handin*. Complete the implementation by Monday and submit as *hwk6*.

2 Notes

It's a good time to practice using test fixtures. If you're going to start using *throw*, then you should start testing for errors. However, for gTest to detect the *throw* we must actually throw something. The easiest, way to do that is to do:

```
throw std::exception();
```

where ever you want to raise an error. You can find documentation on fixtures and testing errors at the these links:

- Fixtures: `https://code.google.com/p/googletest/wiki/Primer#Test_Fixtures:_Using_the_Same_Data_Configuration_for_Multiple_Te`
- Tests for Exceptions: `https://code.google.com/p/googletest/wiki/AdvancedGuide#Exception_Assertions`

To make the *main* procedure code given in the book work with an *enum class* we have to do some explicit type conversion, aka *static_cast*. Basically, an enum class type cannot be automatically converted to and from an integer implicitly. Here's the new code:

```
for(int suit = static_cast<int>(lab7::Suit::CLUBS); suit <= static_cast<int>(lab7::Suit::SPADES); ++sui
  for(int rank = lab7::ACE; rank <= lab7::KING; ++rank){
       std::cout << " " << lab7::Card(rank,static_cast<lab7::Suit>(suit));
  }
}
```