CSE 425 Studio: Object-Oriented Programming Studio I

These studio exercises are intended to give you experience with basic object-oriented language features in C++ as well as to begin exploring logic programming features and abstractions.

In this studio you will again work in self-selected groups of ~3 people, and will report the results of your work on the studio exercises in an e-mail to the course account. Students who are more familiar with the material are encouraged to partner with those who are less familiar with it, and asking questions of your classmates and professor during the studio sessions is highly encouraged as well.

Please record your answers as you work through the following exercises. After you have finished please submit your answers to the required exercises, and to the enrichment exercise if you completed it, by sending e-mail to the course account (cse425@seas.wustl.edu) with "Object-Oriented Programming Studio I" in the subject line. The enrichment exercise is optional, but is a good way to dig into the material a little deeper, especially if you breeze through the required ones.

PART I: REQUIRED EXERCISES

1. Form a team of ~3 people of your choice (preferably a mix of people familiar and unfamiliar with C++) and write down the names of the team members as the answer to this first exercise.

2. Create a new Visual Studio 2013 project for this studio (for example, named **`OOStudioI`** or something similar that identifies which studio this is for). Modify the signature of the main function in your project's main source file (e.g., **`OOStudioI.cpp`**) so that it matches the standard portable (e.g., between Windows and Linux) main function signature (and variable names) for a C++ program: **`int main (int argc, char * argv[])`**.

Add a new header (**`.h`**) and source (**`.cpp`**) file to your project, in which you will declare and define a hierarchy of classes for different kinds of predicate objects. In the new header file, declare a **`Predicate`** base class with a public *abstract* (called *pure virtual* in C++, and indicated by the suffix **`=0`**) **`print`** method that takes no arguments and returns no result, as in:

```
virtual void print () = 0;
```

In that same header file (or in additional header files if you'd like, which is good practice for modularity, though you then need to be sure to put the necessary include statements in various files) declare classes **`UnaryIntegerPredicate`** (for a predicate that takes one integer argument), and **`BinaryIntegerPredicate`** (for a predicate that takes two integer arguments), which inherit (via public inheritance) from the **`Predicate`** base class, and that each override the base class' abstract print method with a public concrete virtual print method, as in:

```
virtual void print ();
```

In the corresponding source file (or source files) define the concrete print method for each class so that it first prints out the input types it takes followed by a colon and then the type it returns. For example the **BinaryIntegerPredicate** class **print** method would print out something like

int, int : bool.

In your program's main function, declare objects of all of the class types including the **Predicate** base class as well as its subclasses (don't forget to include the header file or header files you just wrote in the main source file) and for each of those objects call its print method. Build your program, and see for which of the classes C++ will allow you to declare instances. Comment out any that cannot be instantiated, and then rebuild and run your program. As the answer to this exercise, please say which classes could be instantiated and what output was produced by each of those when you called their print methods.

3. Declare a public pure virtual function call operator in the **BinaryIntegerPredicate** class, which takes two integer arguments and returns a bool value, as in

```
virtual bool operator() (int, int) = 0;
```

Declare and define two subclasses of **BinaryIntegerPredicate**: **LessThanIntegerPredicate** and **EqualsIntegerPredicate**. In each of those subclasses declare and define a concrete function call operator that overrides the pure virtual function call operator that each inherited from its base class. The definition of the concrete function call operator in the **LessThanIntegerPredicate** class should return true if and only if the first integer passed to it is less than the second integer passed to it, while the function call operator in the **EqualsIntegerPredicate** class should return true if and only if the arguments passed to it are equal.

In your main function, declare objects of the **LessThanIntegerPredicate** and **EqualsIntegerPredicate** classes, and call them (via their function call operators) with different integer values that cause them to return different results. Build and run your program, and as the answer to this exercise, please indicate: (1) were you still able to instantiate an object of the **BinaryIntegerPredicate** class and if not why not? (2) what were the different integer values with which you called the new predicates you instantiated, and what result was produced each time?

4. Declare a public pure virtual function call operator in the **UnaryIntegerPredicate** class, which takes one integer argument and returns a **bool** value. Declare and define the following subclasses of the **UnaryIntegerPredicate** class: **ZeroIntegerPredicate**, **NonZeroIntegerPredicate**, **OddIntegerPredicate**, and **EvenIntegerPredicate**. In each of those subclasses, declare and define appropriate concrete virtual function call operators, which override the inherited pure virtual function call operator, and whose behavior corresponds intuitively to the name of the class (e.g., the function call operator for the **ZeroIntegerPredicate** should return true if and only if the value of the passed integer is 0).

In your main function, declare objects of the **ZeroIntegerPredicate**, **NonZeroIntegerPredicate**, **OddIntegerPredicate**, and **EvenIntegerPredicate** classes, and call them (via their function call operators) with different integer values that cause them to return different results. Build and run your program, and as the answer to this exercise, please indicate: (1) were you still able to instantiate an object of the **UnaryIntegerPredicate** class and if not why not? (2) what were the different integer values with which you called the new predicates you instantiated, and what result was produced each time?

5. Declare and define a new **ConstantPredicate** (for a predicate that takes no arguments) subclass of the **Predicate** base class, with an appropriate pure virtual function call operator (which takes no arguments and returns a bool value) and an appropriate concrete print method.

Declare and define a new class called **UnaryIntegerPredicateBinder** that inherits from the **ConstantPredicate** class, and has two private member variables: an integer and a reference (indicated by the **&** symbol) to a **UnaryIntegerPredicate** object. Declare and define a constructor for the **UnaryIntegerPredicateBinder** class that takes an integer and a reference to a **UnaryIntegerPredicate** object, and uses them to initialize its member variables (**hint:** reference member variables must be initialized in a constructor's base/member initialization list, rather than in the body of the constructor, and it's a good idea to initialize all other member variables there as well). Declare and define a concrete function call operator that overrides the inherited pure virtual function call operator with no arguments, which returns the result of calling its **UnaryIntegerPredicate** member variable (via its function call operator) with its integer member variable.

In your main function, construct several **UnaryIntegerPredicateBinder** objects using different integer values and different **UnaryIntegerPredicate** objects, and as the answer to this exercise describe the different combinations you tried and what the results were from calling the **UnaryIntegerPredicateBinder** objects (via their function call operators).

6. Declare and define **TruePredicate** and **FalsePredicate** subclasses of **ConstantPredicate** that return true or false respectively when their function call operator is invoked. In your main function declare instances of these classes, and call the function call operators of the ones that you are allowed to instantiate. As the answer to this exercise please say which you were allowed to instantiate and what the results of calling their function call operators were.

PART II: ENRICHMENT EXERCISE (optional, feel free either to skip or to try variations interest you)

7. Similar to exercise 5, declare and define **BinaryIntegerPredicateFirstBinder** and **BinaryIntegerPredicateSecondBinder** subclasses of the **UnaryIntegerPredicate** class, which can bind either the first or second argument (respectively) of a **BinaryIntegerPredicate** object that is passed via the constructor call. Construct several of those objects using different integer values and different **UnaryIntegerPredicate** objects, and as the answer to this exercise describe what you saw.