CSE687 Midterm #3

Name: _____Instructor's Solution_____

This is a closed book examination. Please place all your books on the floor beside you. You may keep one page of notes on your desktop in addition to this exam package. All examinations will be collected promptly at the end of the class period. Please be prepared to quickly hand in your examination at that time.

If you have any questions, please do not leave your seat. Raise your hand and I will come to your desk to discuss your question. I will answer all questions about the meaning of the wording of any question. I may choose not to answer other questions.

You will find it helpful to review all questions before beginning. All questions are given equal weight for grading, but not all questions have the same difficulty. Therefore, it is very much to your advantage to answer first those questions you believe to be easiest.

1. What is meant by the term scope in the context of a C++ program? What is the purpose of scopes and what happens when your program enters and leaves a scope?

   Answer:

   There are two kinds of scopes. Classes, structs, namespaces, and enums define scopes that have compile-time only effects – to define names and access. Functions and control statements introduce scopes that have runtime effects as well.

   Scopes are blocks of statements defined by curly braces "{" and "}". There are also braceless scopes defined by "if", "else", "for", "while", and "do" statements containing a single executable statement.

   When a runtime scope is entered a stack frame is allocated to hold local data and, for functions, input parameters and a return value. As the execution leaves a scope the destructors for all destructible objects created in that scope are called and the memory is invalidated, e.g., becomes available for execution of the next scope.

2. Write all the code for a parser rule that detects C++ executable statements. Hint: you may wish to eliminate modifiers that don't affect the decision.

   Answer:

```
class Executable : public IRule            // declar ends in semicolon
{                                          // has type, name, modifiers &
public:                                    // initializers.  So eliminate
  bool isModifier(const std::string& tok)  // modifiers and initializers.
  {                                        // If you have two things left
    const size_t numKeys = 12;             // it's declar else executable
    const static std::string keys[numKeys] = {
      "const", "extern", "friend", "mutable", "signed", "static",
      "typedef", "typename", "unsigned", "volatile", "&", "*"
    };
    for (int i = 0; i<numKeys; ++i)
    if (tok == keys[i])
      return true;
    return false;
  }
  bool doTest(ITokCollection*& pTc)
  {
    ITokCollection& tc = *pTc;
    if (tc[tc.length() - 1] == ";")
    {
      // remove modifiers, comments, newlines, returns, and initializers
      SemiExp se;
      for (size_t i = 0; i < tc.length(); ++i)
      {
        if (isModifier(tc[i]))
          continue;
        if (se.isComment(tc[i]) || tc[i] == "\n" || tc[i] == "return")
          continue;
        if (tc[i] == "=" || tc[i] == ";")
          break;
        else
          se.push_back(tc[i]);
      }

      if (se.length() != 2)  // not a declaration so is an executable
      {
        doActions(pTc);
        return true;
      }
    }                          // To make this generally useful you also
    return true;               // need to condense template type tokens
  }                            // and remove invocation parentheses.
};                             // See ActionsAndRules in MTS14 code for
                               // details.
```

3. State the Liskov Substitution Principle (LSP). What code defects break the operation of LSP?

   Answer:

   Any code that uses a base pointer or reference may be bound to an instance of a class derived from the base. The using code need have no knowledge of the type of the bound object or any of its details.

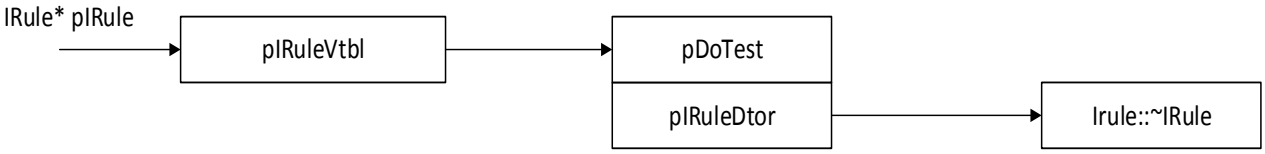
   Liskov Substitution breaks if we:

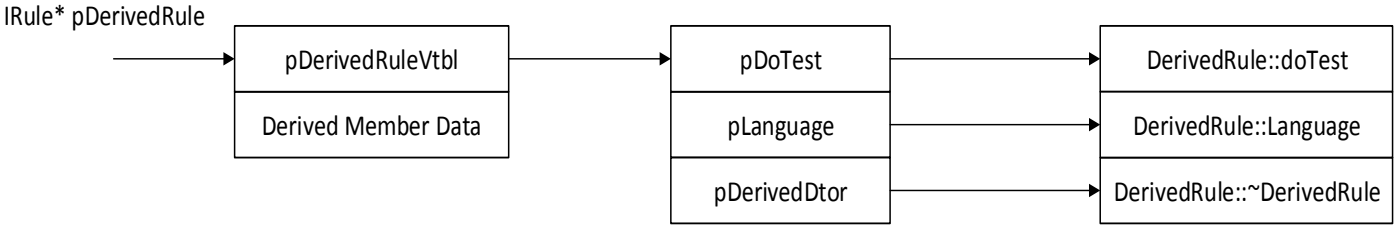
   - Override in the derived class non-virtual functions of the base.
   - Fail to provide a virtual destructor in the base class.
   - Overload a base class function in the derived class.
   - Overload a base virtual function.
   - Use default parameters in both base and derived classes with different values for the default.

4. How many virtual function pointer tables (vtbls) are created for Parser's rules and actions? Draw the vtble for a rule that provides a virtual method void language(), not part of the IRule interface, that supports configuring the rule for either C++ or C#. You are not asked to implement any code for this question.

   Answer:

   There is one vtble for the IRule interface and one for the IAction interface and one for each class derived from those interfaces.





   IRule::addAction and IRule::doActions are not virtual and so do not have entries in the IRule vtbl. IRule::doTest is pure virtual so it has an entry but no corresponding implementation.

   Since DerivedRule::language is virtual, but not part of the IRule interface there is an entry only in the DerivedRule vtble for this function. Note that it cannot be called using an IRule pointer. You have to use a dynamic cast to call it.

5. The std::for_each function from the standard algorithm library is declared like this:

   ```
   template<class InputIterator, class Function>
   for_each(InputIterator first, InputIterator last, Function fn)
   ```

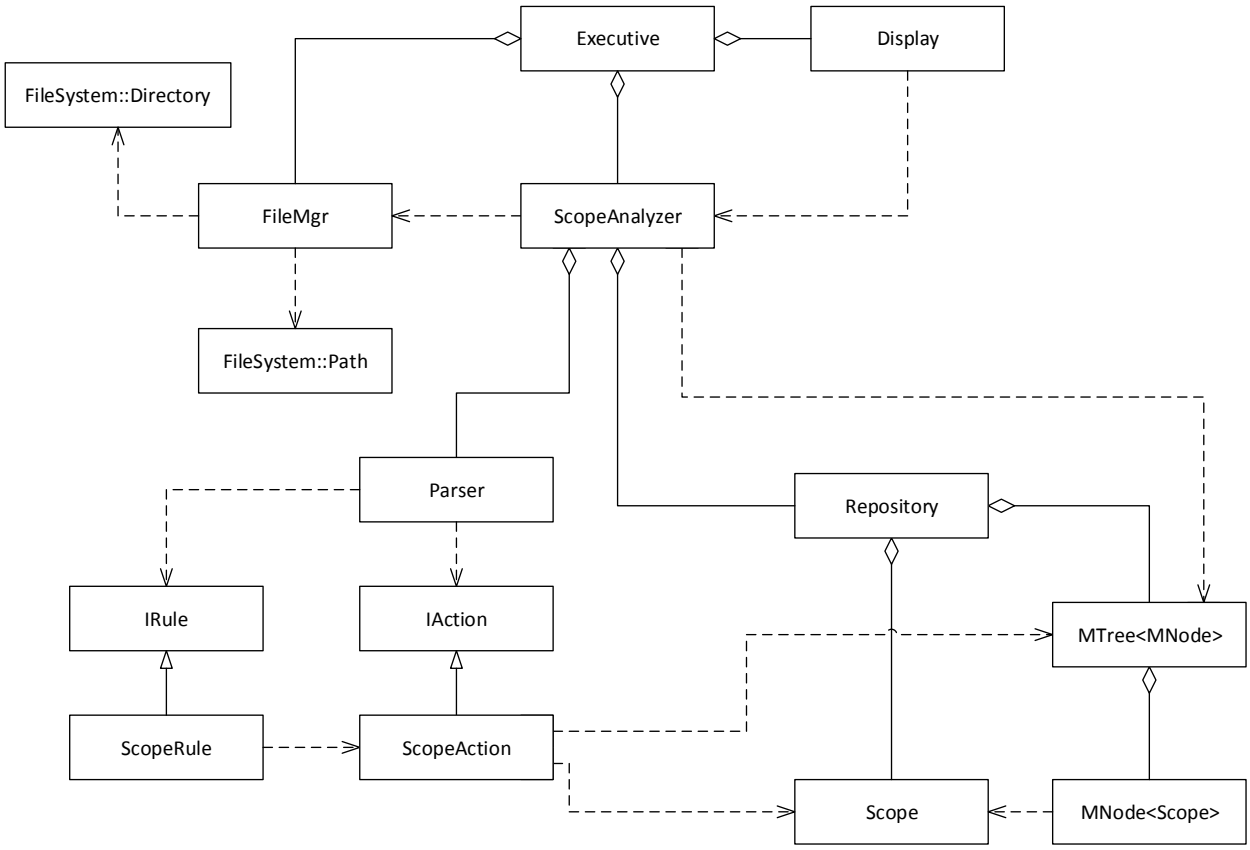
   where fn is a callable object. Write all the code to find the average value of a list of doubles using a lambda as the callable object. Please construct a test list and apply your code to that.

   Answer:

```
std::list<double> ld{ 0.0, 0.5, 1.0, 1.5, 2.0, 2.5 };
double sum = 0;
for_each(begin(ld), end(ld),
  [&sum](double t) {
    sum += t;
  }
);
sum /= ld.size();
```

6. Draw a class diagram for your design of Project #1. You do not need to include classes associated with the parser.

   Answer:



ScopeAnalyzer owns a configParser object, not shown for lack of space, that owns the Parser, Repository, and all Rules and Actions. It gets a file specification from FileMgr and uses Parser to analyze its code scopes. The ScopeAction stores the Scopes in MTree. At the end of analysis ScopeAnalysis returns to Executive which then calls Display to display the required scope information and one tree's XML. XmlWriter has also been elided for lack of space.

7. Write all the code for an object factory that returns a copy of a prototype instance of some unspecified type. Show how you would create and invoke the factory.

   Answer:

```
#include <string>
#include <iostream>

template <typename T>
class PrototypeFactory
{
public:
  T* Create() { return new T(*_pPrototype); }
  void Register(T* pPrototype) { _pPrototype = pPrototype; }
private:
  T* _pPrototype;
};

class TheObject
{
public:
  TheObject(const std::string& str) : _str(str) {}
  void say() { std::cout << "\n  " << _str; }
private:
  std::string _str;
};

int main()
{
  std::cout << "\n  MT3Q7 - Prototype Factory";
  std::cout << "\n ==========================\n";

  TheObject to("Hello CSE687");
  PrototypeFactory<TheObject> protoFact;
  protoFact.Register(&to);

  TheObject* pTO = protoFact.Create();
  pTO->say();
  delete pTO;

  std::cout << "\n\n";
}
```