CSE687 Midterm #4

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 encapsulation? Why is encapsulation important and how do you ensure a class you design is properly encapsulated?

   Answer:

   Encapsulation is the process of implementing a class or function so that its implementation is not accessible to clients of the class. It also means that the design should not require users to know the design details of the class or function.

   Encapsulation prevents breakage of client design when the encapsulated entity's implementation changes.

   We ensure class encapsulation by making all member data private or protected. We also need to make functions that require design knowledge to use correctly to be private or protected.

   Returning pointers and using non-constant reference arguments in class methods weakens encapsulation as does the use of protected data. We occasionally decide to design our classes this way for convenience, but C++11 makes it much easier to avoid at least the use of non-constant references by returning tuples instead of using side-affects to alter the state of more than one external object.

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

Answer:

```
class Declaration : 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;             // its 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)  // has type and name so must be declaration
      {
        doActions(pTc);      // To make this generally useful you will
        return true;         // also need to provide functions to compress
      }                      // template types and to remove parentheses
    }                        // from invocations.  See MT14-Code for
    return true;             // details.
  }
};
```

3. State the Interface Segregation Principle (ISP). What code defects does ISP address? What modifications would you make to a class with a very large interface in order to support ISP?

   Answer:

   The Interface Segregation Principle states that clients should not be forced to depend on class interface elements they do not need to use.

   When interfaces are large many clients may only use a small part of the interface. However, if any part of the interface changed, even parts they do not use, they must be recompiled, as compilation may change the layout of instructions in memory. This isn't a problem for small clients, but when a client is large its compilation may take a significant amount of time and even require many project and environment settings and access to the versions of code on which they depend that are the same as when they were originally compiled.

   For that reason it is important to factor interfaces into cohesive parts where any client is only required to incorporate the part(s) it uses.

   This can be done by factoring the class with a large interface into a core class and additional interfaces that serve specific types of clients. Then for any client a class is derived that inherits from the core base plus other base(s) that provide the specific functionality needed. Supporting interface queries, as in the AbstractProduct demonstration, allows a client to select, at run-time, the interfaces needed.

   It can also be done by factoring the large class into smaller cohesive classes as in Parser where processing is factored into Rules and Actions.

   Finally, template specialization can be used to provide interfaces appropriate for specialized clients.

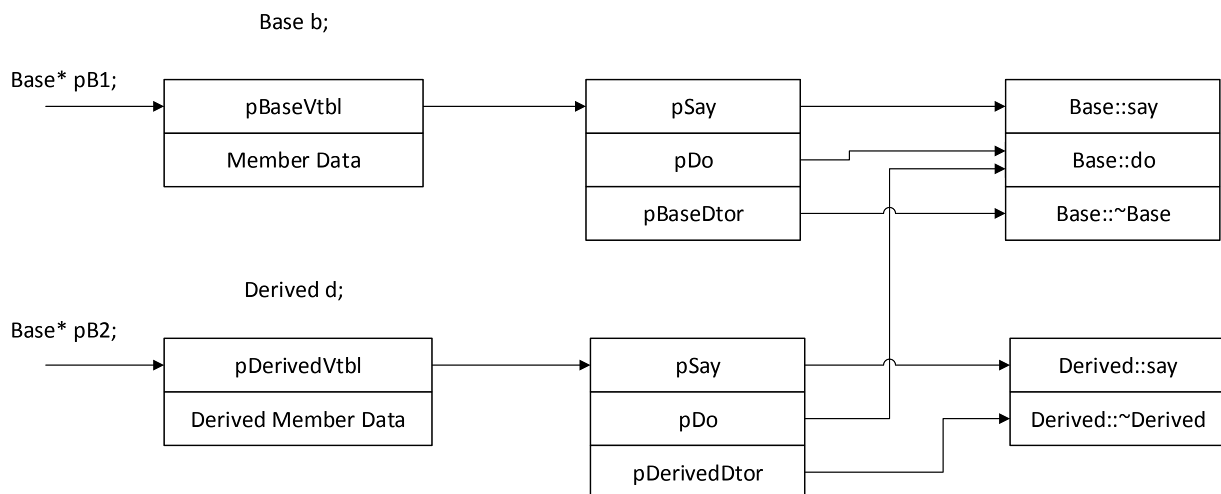
4. What is the purpose of a virtual function pointer table (vtbl). Declare a pair of simple classes which have vtbl(s) and draw a sketch of the table(s).

   Answer:

   The Virtual Function Pointer Table supports dynamically binding functions based on the object called, not the caller. It is what makes Liskov substitution work.

```
class Base
{
  virtual ~Base() {}
  virtual void do();
  virtual void say();
  void see();
};

class Derived : public Base
{
  virtual ~Derived() {}
  virtual void say();
  void taste();
};
```



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 display the value of each element of a collection using a lambda as the callable object. Please construct a test collection (you can choose the type) and apply your code to that. What requirements of the vector items have to be satisfied for this to work? Write a template function that uses your display code where the template parameter is the type of the collection.

   Answer:

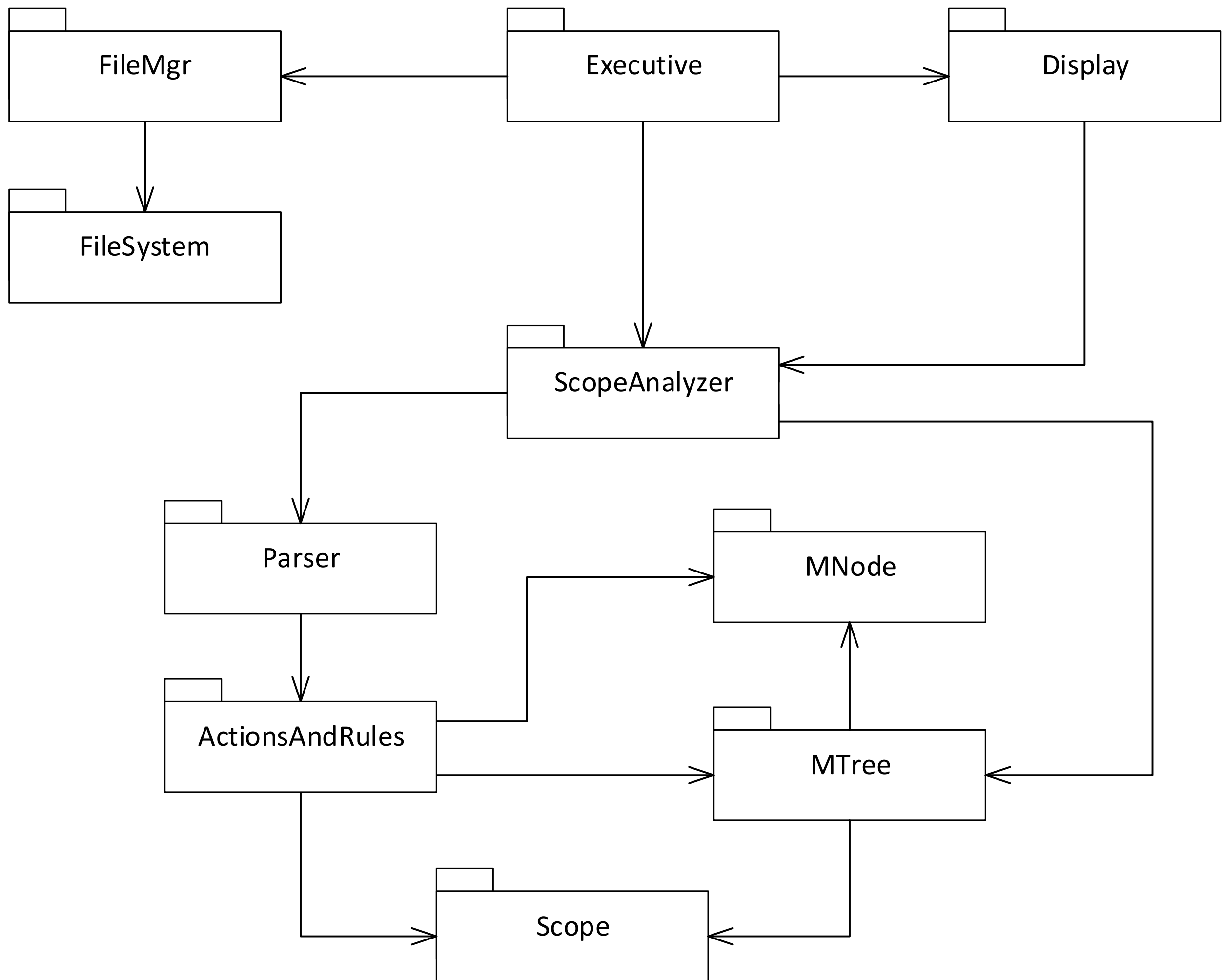
```
template <typename Cont>
void Display(Cont cont)
{
  std::cout << "\n  Display elements of container:";
  std::cout << "\n --------------------------------";
  std::cout << "\n  ";
  bool first = true;
  for_each(begin(cont), end(cont),
    [&first](Cont::value_type t) {
      if (!first)
      {
        std::cout << ", " << t;
      }
      else
      {
        std::cout << t;
        first = false;
      }
    }
  );
  std::cout << "\n";
}
```

```
Cont::value_type must support insertion with operator<<.

See MT14-Code MT4Q5 for rest of the code that satisfies remaining
requirements of this question.
```

6. Draw a package diagram for your design of Project #1. You do not need to include packages used by the parser for scanning and building the parser.

   Answer:



ScopeAnalyzer owns an instance of ConfigParser which owns Parser, derived classes from ActionsAndRules, and a Repository. The Repository owns an MTree which a derived action to store its acquired scope information. When analysis is complete the ScopeAnalyzer returns to Executive which requests Display to show the analysis results.

7. A Symbol Table holds information about types encountered in code you are analyzing. Please design a symbol table in which each record holds a type name, an instance name, and some additional information wrapped in an instance of some class that is defined by the user of the Symbol Table. Make your table hold information that is relevant to your Project #2 design. How will you find specific records in the table?

   Answer:

```
template <typename TypeInfo>
class SymbolTable
{
public:
   using Type = std::string;
   using Name = std::string;
   using Record = std::tuple<Type, Name, TypeInfo>;
   using Records = std::vector<Record>;

   void add(const Record& record) { _records.push_back(record); }
   Record& operator[](size_t i);
   Record operator[](size_t i) const;
   Records FindName(const Name& name);
   Records FindType(const Type& type);
   Records GetRecords() { return _records;  }

private:
   Records _records;
};

struct Type_Info
{
   using StartLine = size_t;
   using EndLine = size_t;
   using Complexity = size_t;
   using ChildTypes = std::vector<std::string>;

   StartLine startLine;
   EndLine endLine;
   Complexity complexity;
   ChildTypes childTypes;
};
```