CSE687 Midterm #4

Name: ____Instructor's Solution_____ SUID: _____

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 Exams 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 the easiest.

1. How do you declare that a class method will not change the state of instances of the class? When will a program fail to compile if you don't use that declaration?

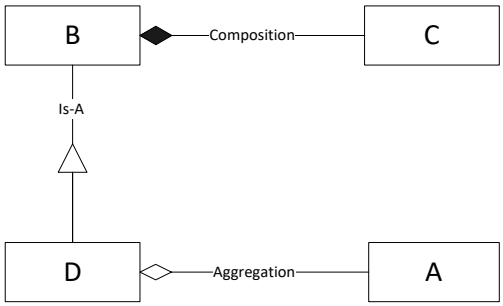
   Answer:

   You declare the class method with the keyword const following the closing parenthesis, i.e.:

```
class A {
          someMemberFunction() const { … }
          otherMemberFunction(const B& b) { … }
        };
class B { … };
A a;
a.someMemberFunction();  // this method won't change state of a
B b;
a.otherMemberFunction(b);  // this method won't change state of b but may change a
```

   If you have passed, to some function, an instance of the class as a const, const*, or const&, and attempt to call a member function on that instance, compilation will fail unless the method is qualified as const.

2. Given the compound object described in the diagram, write a copy constructor for the class D, assuming that classes B and C have correct copy construction semantics.



Answer:

This answer assumes that the D class has a member std::string msg_; An answer can be entirely correct without this member.

```
//----< copy construction of D >-------------------------------------

D::D(const D& d) : B(d), msg_(d.msg_)
{
  std::cout << "\n  copy construction of D";
  if (d.pA)
    pA = new A(*d.pA);
  else
    pA = nullptr;
}
```

Note: B copy constructor takes care of copying C, so D copy constructor simply calls B(d).

3. State the Liskov Substitution Principle (LSP). What code constructs will cause LSP to fail?

   Answer:

   Functions that use pointers or references statically typed to some base class must be able to use objects of classes derived from the base through those pointers or references without any knowledge specialized to the derived classes.

   LSP fails if your code:
   a. Overloads across class scopes (error of the derived class)
   b. Overloads virtual functions (error of the base class)
   c. Redefining non-virtual member functions of the base in a class derived from the base
   d. Failing to provide a virtual destructor in a class with at least one virtual function

   Designing a derived class that has stronger pre-conditions or weaker post-conditions on its methods is a design error, but not a code construct error.

4. What relationships can you use between classes? Describe where you have used each of them in your Project #2. Please be specific.

   Answer:

   The Object Oriented class relationships are inheritance, composition, aggregation, and using relationships.

   In Project #2:
   a. Rules derive from IRule, Actions derive from IAction.
   b. Executive composes DepAnal.
   c. ConfigureParser aggregates all the derived rules and actions, parser, semiExp, and Toker.
   d. DepAnal uses the Abstract Syntax Tree, ScopeStack, and Repository.

5. Write code for a class that provides thread-safe insertion, retrieval, and deletion of elements in a std::unordered_map<std::string, std::vector<std::string>>. Please provide for insertion and extraction.

   Answer:

   I have included all of the obviously useful members of ThreadSafeMap. You are only required to provide the equivalent of ThreadSafeMap::insert and ThreadSafeMap::find.

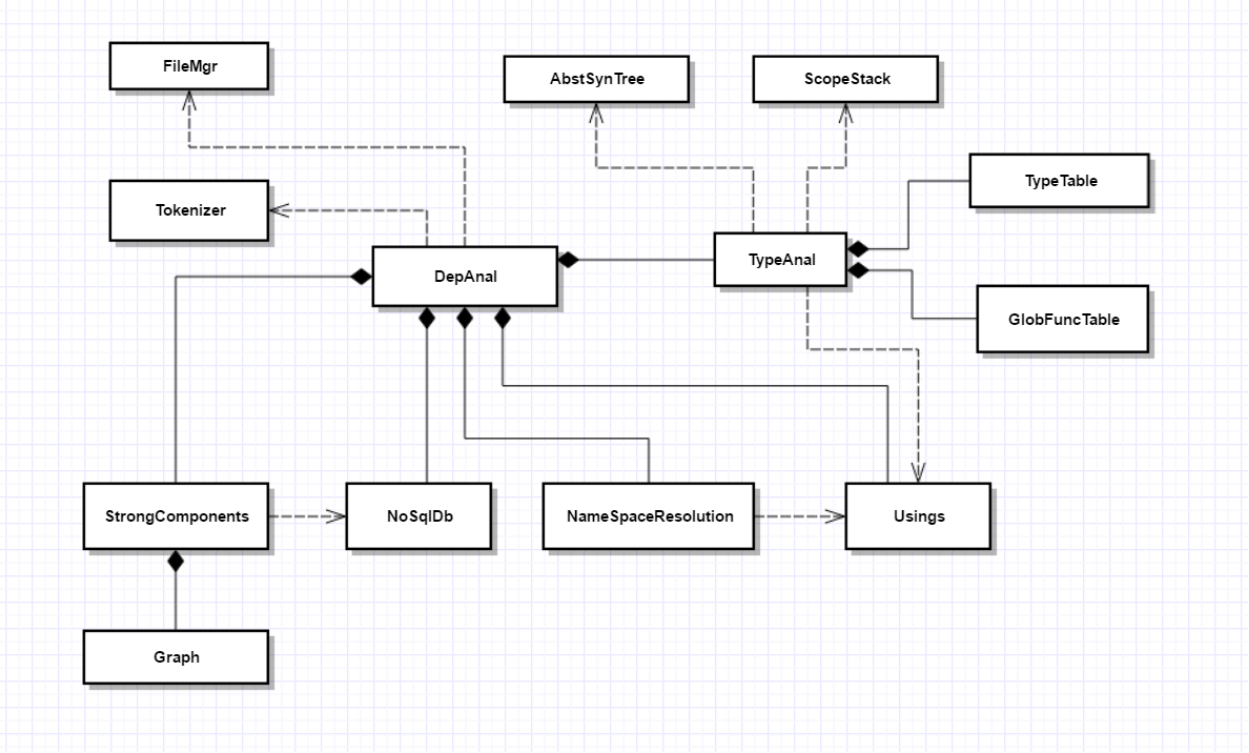
```
class ThreadSafeMap
{
public:
  using Key = std::string;
  using Value = std::vector<Key>;
  using Map = std::unordered_map<Key, Value>;
  using iterator = Map::iterator;

  bool hasKey(const Key& key);
  bool insert(const Key& key, const Value& value);
  Value find(const Key& key);
  bool set(const Key& key, const Value& value);
  bool remove(const Key& key);
  void clear();
  iterator begin();
  iterator end();
  std::recursive_mutex& getLock() { return mtx_;  }
private:
  Map map_;
  std::recursive_mutex mtx_;
};

inline bool ThreadSafeMap::hasKey(const Key& key)
{
  std::lock_guard<std::recursive_mutex> lck(mtx_);
  if (map_.find(key) != map_.end())
    return true;
  return false;
}

inline bool ThreadSafeMap::insert(const Key& key, const Value& value)
{
  std::lock_guard<std::recursive_mutex> lck(mtx_);
  if (hasKey(key))
    return false;
  map_[key] = value;
  return true;
}

inline ThreadSafeMap::Value ThreadSafeMap::find(const Key& key)
{
  std::lock_guard<std::recursive_mutex> lck(mtx_);
  if (map_.find(key) == map_.end())
  {
    Value retVal;  // empty collection
    return retVal;
  }
```

```
  return map_[key];
}

inline bool ThreadSafeMap::set(const Key& key, const Value& value)
{
  std::lock_guard<std::recursive_mutex> lck(mtx_);
  if (map_.find(key) == map_.end())
    return false;
  map_[key] = value;
  return true;
}

inline bool ThreadSafeMap::remove(const Key& key)
{
  std::lock_guard<std::recursive_mutex> lck(mtx_);
  if (map_.find(key) == map_.end())
    return false;
  map_.erase(key);
  return true;
}

inline void ThreadSafeMap::clear()
{
  std::lock_guard<std::recursive_mutex> lck(mtx_);
  map_.clear();
}

inline ThreadSafeMap::iterator ThreadSafeMap::begin()
{
  std::lock_guard<std::recursive_mutex> lck(mtx_);
  return map_.begin();
}

inline ThreadSafeMap::iterator ThreadSafeMap::end()
{
  std::lock_guard<std::recursive_mutex> lck(mtx_);
  return map_.end();
}
```

Note: static mutexes in each function do not ensure thread-safety. One thread may be inserting while another is finding, which results in each thread using a different mutex, e.g., no locking.

6. Draw a class diagram for your implementation of dependency analysis for Project #2. You don't need to include any of the classes provided in helper code.

   Answer:



7. Write all the code for a function that displays the Run-time Type Id of an instance of an unspecified type.

   Answer:

   You may either push to std::cout in the function or return a string to be used by caller.

```
template<typename T>
std::string typeOf(const T& t)
{
  return typeid(t).name();
}

int main()
{
  class Widget {};

  std::cout << "\n  typeOf(\"a string\") returns \""
            << typeOf("a string") << "\"";
  std::cout << "\n  typeOf(std::string(\"another string\")) returns \""
            << typeOf(std::string("another string")) << "\"";
  std::cout << "\n  typeOf(Widget) returns \"" << typeOf(Widget()) << "\"";
  std::cout << "\n\n";
}
```