CSE687 Midterm #1

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. Describe what happens when a C++ exception is thrown in library code that your program is calling. There are several cases to consider, depending on how your code is structured.

   Answer:

   If there is no enclosing try block the exception is unhandled and terminate() is called which, by default, calls abort().

   If there is an enclosing try block, the exception is matched against each declared catch clause exception type. If any match, all objects that have been created from the throw site up to the matching try block are destroyed, with the exception of the exception object[1]. Then the handler executes, and execution resumes with the statement following the last catch clause for that try block.

   If none of the catch clause types match the exception, it navigates to the next enclosing try block, if any, and the exception is matched with each of the catch clause exceptions. If a match is found the existing objects from the throw site up to the matching try block are destroyed, and the matching catch handler executes.

   If catch clause matching succeeds for any enclosing try scope, the exception is handled and execution resumes following the last catch clause for the matching try scope. If none succeeds, the exception is unhandled, and terminate() is called.

   If a second exception is thrown before the original is handled, terminate() is called.

   Note that exceptions must be thrown by value and should be caught be reference.

[1] This process is called stack unwinding

2. State the Liskov Substitution Principle (LSP). How does the C++ language support LSP?

   Answer:

   Any function accepting a pointer or reference to an instance of a base type will accept a pointer or reference to a class derived from the base. The function's use of the pointer or reference will not depend on any knowledge specific to the derived class.

   C++ has two features essential to Liskov Substitution.

   a. All inheritable functions[2] of the base class are inherited by every derived class. That means that any calls made by a function accepting a base class pointer or reference will succeed with a pointer or reference to an instance of a class derived from the base.
   b. Each class with at least one virtual function has a Virtual Function Pointer Table (VFPT). That table directs calls, to virtual functions that have not been overridden, to the base function code. Any call to a virtual function that has been overridden will be directed to the derived class code, by a dispatching process that uses the VFPT. It is the VFPT that allows a derived class to specialize the behavior of its base.

[2] Constructors, destructor, and assignment are not inherited, but will be created by the compiler if needed. They have the same syntax in both base and derived, so substitutability is preserved.

3. Write all the code for a class that saves namespace using information for files[3]. How would you analyze code to decide what goes into the table?

   Answer:
   For each file detect, with a Parser Rule, using namespace declarations, and add to the table with the file name as key. Also, detect all namespace qualifications in the file's code, and add to the table.

```
class Usings
{
public:
  using File = std::string;
  using Namespace = std::string;
  using Using = std::vector<Namespace>;

  bool hasFile(File file);
  void insert(File file, Namespace nmspace);
  Using usings(File file);
  void clear();
private:
  std::unordered_map<File, Using> uMap;
};

bool Usings::hasFile(File file)
{
  if (uMap.find(file) != uMap.end())
  {
    return true;
  }
  return false;
}

void Usings::insert(File file, Namespace nmspace)
{
  if (hasFile(file))
  {
    uMap[file].push_back(nmspace);
    return;
  }
  Using uses;
  uses.push_back(nmspace);
  uMap[file] = uses;
}

Usings::Using Usings::usings(File file)
{
  if (hasFile(file))
    return uMap[file];
  return Using();
}

void Usings::clear()
{
  uMap.clear();
}
```

In order to resolve namespace qualifications you need to know what namespaces a file uses.

With that information you can find the depended upon file from a TypeTable with value containing a collection of file:namespace pairs.

Using Namespace Information:

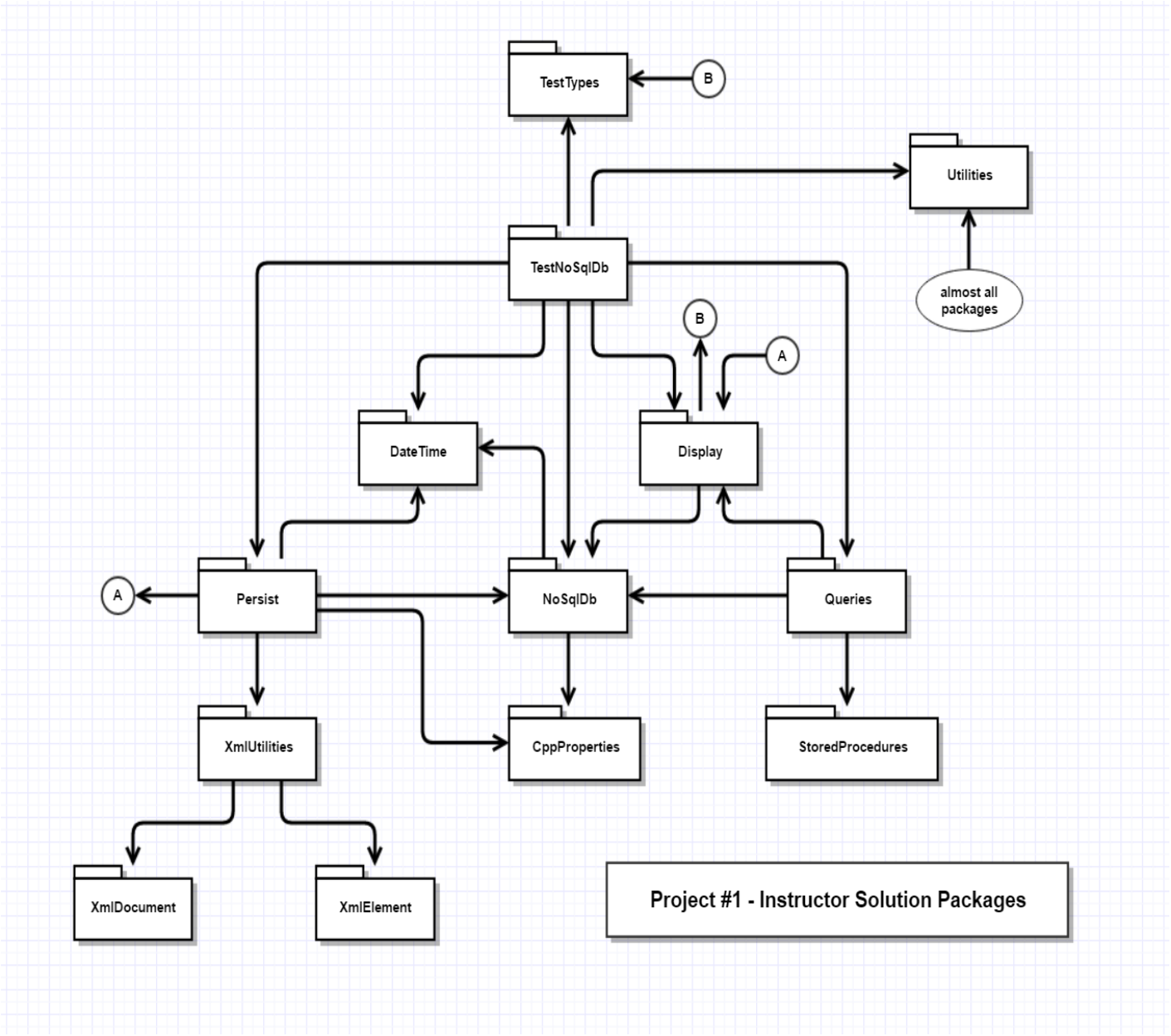
File1 : ns1, ns2, …

File2 : ns3, ns4, …

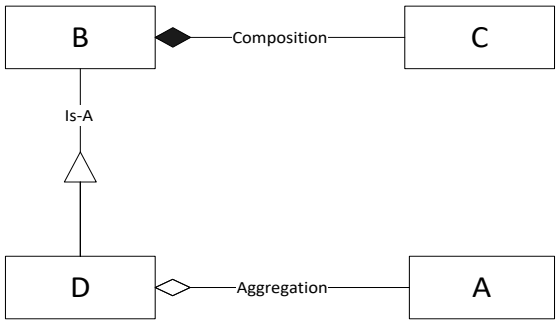
Grading note:
This class is not analyzing anything. It simply STORES namespace using information.

[3] Each source code file many have (a small number of) using statements and may explicitly use namespace::type.

4. Draw a package diagram for your implementation of Project #1.



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




```
The code below assumes that D contains a std::string member msg_.  Your
answer does not need to include that.


//----< move construction of D >-------------------------------------

D::D(D&& d) : B(std::move(d)), msg_(std::move(d.msg_)), pA(d.pA)
{
  std::cout << "\n  move construction of D";
  d.pA = nullptr;
}
```

Grading Note: It's up to B's constructor to handle C appropriately.

6. Describe several ways to pass arguments to a thread you are creating and to retrieve results from the thread. You may use code in your discussion, but are not required to do so.

   Answer:

   You can pass arguments to a thread in the following ways:

   a. Add each argument to a thread constructor statement, following the thread's callable object, and separated by commas.
   b. You can do exactly the same thing using the async template function.
   c. You can pass the thread a lambda with the arguments it needs as captured variables.
   d. You can pass a functor instance to the thread with the arguments it needs as member data, perhaps passed into the functor's construction statement.
   e. Use shared BlockingQueue to pass a sequence of inputs.

   You can retrieve results from the thread in the following ways:

   f. Use the async template function that returns a future<T> where T is the type of the return value. You eventually call the future's get() function which will block until the answer is ready.
   g. You can pass, using std::ref() a container for the result and pass by std::ref() a std::mutex that the thread function uses when it computes the result, and which the using code acquires before accessing the result. Note that the std::ref()s are required because arguments passed to a thread are passed by value.
   h. Use a shared BlockingQueue to return a sequence of results.
   i. Use a callback to deposit result in some thread-safe location.

   MT3Q4b.cpp shows another way to retrieve results from child thread. That code is nearly equivalent to a custom promise<T>, future<T> implementation.

7. Write all the code for a lambda that sends, to std::cout, a specified string, followed on the next line with a string composed of hyphen characters, e.g., '-'. Show how to execute the lambda.

   Answer:

```
std::function<void(std::string)> title = [](const std::string& msg) {
  std::cout << "\n  " << msg;
  std::cout << "\n " << std::string(msg.length() + 2, '-');
};

int main()
{
  title("MT1Q7 - create a lambda");
  std::cout << "\n  That's the demo folks!";
  std::cout << "\n\n";
}
```