Grammar Issues for CSE687 Project #2[1]

In Project #2 – Hierarchy, you are required to discover the class relationships in and among classes in a set of C++ source code files. Specifically, you must find these relationships:

1. **Inheritance:**
   A class may inherit from another class, a struct, or an interface, using the syntax:

   class myClass : [public | protected | private] yourClass { … }

   which says that myClass inherits from (is derived from) yourClass.

2. **Composition:**
   One class or struct, say A, is composed with another, say B, when B holds an instance or a reference to A as a data member, and the lifetime of A is the same as the lifetime of B. That happens only when A is initialized or constructed in the constructor of B.

```
class B  // case #1 – A is initialized in B's constructor
{
  private:
      A myA;
  public:
      B(args) [: myA(a ε args)] { … }
        // will use default initialization if option is not present
}

class B  // case #2 – A is constructed in B's constructor call
{
  private:
      A[* | &] myA;
  public:
      B(args) { myA = [ |*]new A(a ε args); }
      ~B() { delete [ |&] myA; }  // [option]
      :
}
```

[1] The class text (C++ Programming Language, Stroustrup) has quite complete descriptions of the language grammar spread over many chapters. A very nice condensation with syntax diagrams can be found in:
C++ Complete : A Reference and Tutorial to the Proposed C++ Standard, Anthony Rudd, Wiley, 1994.
Note that you need much, much, much less than a complete specification for the language grammar, as described in this note.

3. **Aggregation:**
   One class or struct, say A, is aggregated with another, say B, when either B holds a reference to A which is initialized in a non-constructor member function, so that its lifetime is less than the lifetime of the aggregator, B, or when it is constructed as a local data in a member function of B.

```
class B  // case #1 – lifetime of A starts later than B
{
  private:
        A* pA;
  public:
        B() { … }
        someFunction() { pA = new A(); … }
           // correct design will provide for deletion at some point
}

class B  // case #2 – lifetime of A is shorter than B on both ends
{
  public:
        B() { … }
        someFunction() { A myA; … // A destroyed at end of func. }
        :
}
```

4. **Uses:**
   A using relationship occurs when one class, Say A, is accessed by another, say B, but B does not construct the instance of A.

```
class B  // case #1 – B holds onto reference to A for use later
{
  private:
        A* pA;
  public:
        B() { … }
        void anotherFunction(A* pYourA)
        {
           pA = pYourA;
           // use YourA instance in some way here or elsewhere
        }
}
```

5. **Note about declarations:**
   In items #2 and #3 you have to be able to detect declarations of an instance of some type. That is, you want to know that a declaration occurred and what type was declared.

   That can be messy since declarations can use modifiers like const, static, volatile, … , the prefix operator * to indicate a pointer, and the postfix operators [] and = used for defining an initialization sequence.

   You can simplify that by throwing away all modifiers (look up the C++ keywords and decide which may be declaration modifiers), the * prefix if it occurs, and the postfix [] and the postfix = and everything that follows it. You will then be left with two tokens, a Type and its instance identifier.