Benjamin Fenster
CS-100 (Damon)
19 March 2003

Notes – C++

I. Compare and Contrast
   a. Almost everything from CS-100 is language independent (among the object-oriented languages)
   b. Languages have different levels of support for some things.
      i. May need to hand-code / hand-check some elements.
      ii. Some features may be worth avoiding.

II. Exceptions
   a. Anything can be thrown in C++
      i. In Java, only classes that implement Throwable can be thrown.
      ii. In C++, `throw 7;` is valid.
      iii. There's no way to differentiate between two strings (or two integers, or whatever) so you'd have to catch everything or catch nothing.
      iv. Better to use objects, as is required in Java.
   b. No finally clause in C++
      i. Be careful setting locks, starting transactions, et cetera.
      ii. You need to guarantee that every path out of the code will release those locks, commit or rollback those transactions, or do whatever else the `finally` clause would have done.
      iii. That means you need to find every single path and make sure the "whatever" gets done there.
   c. C++ does allow declarations of what's to be thrown, but doesn't require it.
      i. Completely optional.
      ii. Say `throw`, not `throws`.
      iii. Not having a throw clause does NOT necessarily mean nothing will be thrown! It may mean the programmer simply didn't include it.
      iv. To specifically indicate that no exceptions will be raised, say `throw ();`
   d. Two special functions get called in C++ if bad things happen
      i. unexpected()
         1. If you use the throw clause and then throw something not specified, this gets called.
         2. By default, it prints an error and exits.
         3. This can be overridden (globally) to do any necessary cleanup work (then perhaps throw a different exception or whatever)
      ii. terminate()
         1. Called whenever an exception goes un-handled or something is thrown that's not in the throws clause.
         2. There are other weird ways for this to get called that relate to intricacies of the language, but which aren't worth considering here.

III. Aggregate Classes
   a. Part of the Standard Template Library (STL)
   b. Two types of containers
      i. Sequences
         1. Vector. Default.
         2. List. Good for inserts / deletes in the middle.
         3. Deque. Good for inserts / deletes at the ends.
      ii. Associative
         1. Set. Same as in Java
         2. Multiset. Same as a Bag (can have duplicate elements)
         3. Map. As expected.
         4. Multimap. Duplicate keys allowed.
   c. Containers declare what they contain.
      i. Required!
      ii. `list<int> *myList;`

    iii. The Good
        1. That means all the casting on calls to next() and such can be eliminated.
        2. No runtime errors can arise from incorrect casting.
        3. More is checked by the compiler.
    iv. The Bad
        1. Can't have generic code hat handles any ol' list.
        2. Must have separate functions for each type of data you may need to handle.
        3. Makes for extremely large code!
        4. Size of code grows geometrically.

d. Iterators are very different.
    i. `list<int> l;`
    ii. `list<int>::iterator liter;`
    iii. `for (liter=l.begin; liter != l.end(); liter++) *iter=0;`
    iv. The iterator doesn't know its own endpoint so if you don't know with what list it's associated you need to ask it.
e. There is no top-level Object
    i. This shows up everywhere but especially with containers.
    ii. Containers expect certain methods (overloaded operators to be present).
    iii. If you didn't write the class you may not be able to use it with a container.
f. C++ stores embedded objects as wella s pointers / references.
    i. See CS100-23-20, CS100-23-21
    ii. Embedding objects directly can cause problems.
    iii. Subclass may add some storage, which makes it bigger.
    iv. Can't have a Vector of a superclass and insert a subclass directly because it won't fit.
    v. Always, always, always use pointers.
g. Sequences
    i. There's no common superclass to Vector, List, Deque.
    ii. That means you have to choose which one you want to use right away and stick with it.
    iii. Again, there's now way to write generic code.
    iv. All this is demanded by the way STL is implemented.