**Project #1 – Type Dependency Analysis** due Tuesday, February 12  
version 1.1   
  
Purpose:

This project requires you to build or reuse a template-based facility to represent graph data structures. You will use that facility to store type dependency relationships within a specified file set. Type dependencies are due to: inheritance, composition, aggregation, and using relationships, which we discuss in class.

You will find a Parser facility provided here: <http://www.lcs.syr.edu/faculty/fawcett/handouts/CSE687/code/Parser/>, to be useful for analyzing types and type relationships, found in existing files, as required for this project.

Requirements:

Your type dependency project:

1. **shall** use standard C++[[1]](#footnote-1) and the standard library, compile and link from the command line, using Visual Studio 2012, as provided in the ECS clusters and operate in the environment provided there[[2]](#footnote-2).
2. **shall** use services of the C++ std::iostream library for all input and output to and from the user’s console and C++ operator new and delete for all dynamic memory management.
3. **shall** use a template-based graph library to represent type dependencies[[3]](#footnote-3) within a specified file set. This facility **shall** be composed of a graph class which uses the services of a vertex class, in an adjacency structure[[4]](#footnote-4). Each graph vertex has an entry in the adjacency structure. Edges are simply references that a vertex holds to other vertices. Each vertex is accompanied by an instance of a Vertex type. Each vertex **shall** hold a std::vector of std::pairs. Each pair holds a reference to a child vertex and an instance of an Edge type[[5]](#footnote-5).
4. The graph class **shall** provide correct assignment and copy construction, and shall provide Depth First Search (DFS) for graph traversal. This search **shall** accept a function or functor operation, to be executed on each vertex and/or edge of the graph. Your project **shall** provide a functor that enunciates each vertex visited and each edge traversed.
5. The graph class **shall** support containment of an instance of a parametrized type in both graph vertices and edges that join vertices, as required by 3, above. Vertices and edges will each have their own template parameter, so that different types may be so contained. That is, the graph class will be parameterized on both types, like so:   
     
    template <typename VertexType, typename EdgeType> class graph { … };
6. **shall** provide means of finding all vertices or edges that hold a specified value.
7. The graph package **shall** provide two global functions, one that creates an XML representation[[6]](#footnote-6) for a specified graph and another that reads an XML graph representation and creates a graph instance.
8. **shall** provide provide facilities to find all the types defined within a specified file set and analyze their inter-relationships. These will be stored in a dependency graph and written to a file holding its graph representation. File sets are specified with a path and pattern on the command line and an option \R to indicate that the entire directory tree rooted at the path is to be searched recursively.
9. **shall** provide a test executive package and a display package, that, combined with the graph facility, demonstrates you meet all the requirements of this specification.
10. Your project submission **shall** be uploaded in a zip file archive, including two batch files named compile.bat and run.bat that compile your project and run it using appropriate command line arguments. Please also include a Visual Studio solution that when run demonstrates you meet these requirements.

Note that requirement #10 does not ask you to provide a graphical representation of the dependency relationships. You may simply provide a text representation (please design this yourself).

You should think carefully about the output of this program. The quality of your design is measured, in part, by how well you compose the structure of your output. Note that there is no requirement to provide a graphical user interface. This tool can be implemented very effectively with a command-line input and file and console outputs.

The diagrams below illustrate the relationships between the graph structure and the adjacency and vertex structures.



1. This means, for example that you may not use the .Net managed extensions to C++. [↑](#footnote-ref-1)
2. VC++ 2012 is available in all the ECS clusters. [↑](#footnote-ref-2)
3. You may build this or use the graph facility provided here: <http://www.lcs.syr.edu/faculty/fawcett/handouts/CSE687/code/Graph/>. You may not use other graph libraries from the web. [↑](#footnote-ref-3)
4. We will discuss adjacency collection in class. [↑](#footnote-ref-4)
5. We have intentionally been somewhat vague about the nature of an edge reference. This will be discussed carefully in class. Note that the details of this requirement preclude you from using a separate edge class in this project. [↑](#footnote-ref-5)
6. We will discuss formats for the XML file in class. [↑](#footnote-ref-6)