**Project #3 - Parallel Dependency Analysis**

Version 1.0, Revised: 02/13/2016 14:48:18  
Due Date: Tuesday April 5th

**Purpose:**

The first two projects this Spring focused on building software tools for code analysis. We emphasize C++ code but want our tools to be easily extendable to other similar languages like C# and Java.

Code analysis consists of extracting lexical content from source code files, analyzing the code's syntax from its lexical content, and building an Abstract Syntax Tree (AST) that holds the results of our analysis. It is then fairly easy to build several backends that can do further analyses on the AST to construct code metrics, search for particular constructs, or some other interesting features of the code.

You will find it useful to look at the [Parsing](http://www.ecs.syr.edu/faculty/fawcett/handouts/Webpages/BlogParser.htm) blog for a brief introduction to parsing and code analysis.

In this third project we will build and test a type-based dependency analyzer that uses the analysis machinery we developed in the first two projects and exercises the new thread facilities provided in the C++11 standard, as implemented in Visual Studio 2015. We will add, to the packages from Projects #1 and #2, the new packages:

* **ThreadPool1:**

Provides facilities to enqueue work items and execute them asynchronously. Each work item will be a callable object2 that is supplied by application code.

* **Tasks:**

Provides a Task class that executes a specified callable object on a ThreadPool thread, using services of the ThreadPool package.

* **TypeAnalysis:**

Finds all the types defined in each of a collection of C++ source files. It does this by building rules to detect type definitions: classes, structs, enums, typedefs, and aliases, and capture their fully qualified names and files where they are defined.

* **DependencyAnalysis:**

Finds, for each file in a specified file collection, all other files from the file collection on which they depend. File A depends on file B, if and only if, it uses the name of any type or global function defined in file B. It might do that by calling a function or method of a type or by inheriting the type. Note that this intentionally does not record dependencies of a file on files outside the file set, e.g., language and platform libraries.

* **ParallelDependencyAnalysis:**

Evaluates the dependency graph for all the packages in a specified file collection. For each file the analyses run asynchronously, using the facilities of the Task class. I expect you will find it easiest to implement this in two passes over the file collection. In the first pass we find, asynchronously, all the types defined in the collection using services of the TypeAnalysis package. In the second pass we find, asynchronously, all the dependencies between files using the DependencyAnalysis package.

* **ParallelDependencyExecutive3:**

Collects all of the files matching one or more patterns in a directory tree rooted at a specified path. On that collection it conducts a dependency analysis using the services of the ParallelDependencyAnalysis package. It then then displays the results by listing each of the files and there dependencies on two adjacent lines (possibly wrapped).

**Requirements:**

Your DependencyAnalyzer Solution:

1. **Shall** use Visual Studio 2015 and its C++ Windows Console Projects, as provided in the ECS computer labs.
2. **Shall** use the C++ standard library's streams for all I/O and new and delete for all heap-based memory management4.
3. (**2**) **Shall** provide C++ packages as described in the **Purpose** section. You are free to alter the names and division of responsibilities as you see fit. However, you are expected to enforce the "Single Responsiblity Principle"5 as illustrated in the package structure described above.
4. (**3**) **Shall** provide a ThreadPool package that declares and defines a ThreadPool class that executes enqueued work items asynchronously6.
5. (**2**) **Shall** implement a Task package that declares and defines a Task class that executes a callable object on a ThreadPool thread.
6. (**4**) **Shall** provide a TypeAnalysis package that identifies all of the types defined in a specified file. It is expected that you will use the analysis machinery you developed in Projects #1 and #2.
7. (**3**) **Shall** Provide a DependencyAnalysis package that identifies all of the dependencies between files in a specified file collection.
8. (**3**) **Shall** provide ParallelDependencyAnalysis package that finds compilation dependencies between all of the files in a specified file collection using the asynchronous processing facilities provided by the Task package.
9. (**3**) **Shall** provide a ParallelDependencyExecutive that finds all the compilation dependencies between files in a file collection. Each of the files in the collection are found in a directory tree rooted at a specified path and whose names match one or more specified patterns.
10. (**5**) **Shall** include an automated unit test suite that demonstrates you meet all the requirements of this project7.
11. We will discuss the design of the ThreadPool class. You may, if you wish, elect to wrap the Windows ThreadPool in the ThreadPool class rather than implement it yourself.
12. These work item callable objects are likely to be implemented with C++ lambdas bound to std::function objects.
13. I expect that you will need to use the FileMgr package from Project #2 and also develope a Display package. You may discover the need for other low-level packages as well.
14. That means that you are not allowed to use any of the C language I/0, e.g., printf, scanf, etc, nor the C memory management, e.g., calloc, malloc, or free.
15. <https://en.wikipedia.org/wiki/Single_responsibility_principle>
16. This will be discussed in class.
17. This is in addition to the construction tests you include as part of every package you submit.