Project #2 - Type-based Dependency Analysis

Version 2.1, Revised: 02/06/2017 13:49:20
Due Date: Tuesday March 7th

Purpose:

The second project this Spring focuses 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 a Type Table (TT) or an Abstract Syntax Tree (AST) that hold the results of our analysis. It is then fairly easy to build several backends that can do further analyses on the data held in TT or 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 blog for a brief introduction to parsing and code analysis.

In this second project we will build and test a type-based dependency analyzer that uses the analysis machinery I developed for the CodeAnalyzer that the TAs and I run on your projects. You will add to the packages from the CodeAnalyzer, the new packages:

- **TypeTable:**
  Provides a container class that stores type information needed for dependency analysis.
- **TypeAnalysis:**
  Finds all the types and global functions 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.
  - global function definitions
  - global data definitions

  and capture their fully qualified names and the files where they are defined. It uses the TypeTable package to store that information.
- **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 or global data 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.
- **Strong Components:**
  A strong component is the largest set of files that are all mutually dependent. That is, all the files which can be reached from any other file in the set by following direct or transitive dependency links. The term "Strong Component" comes from the theory of directed graphs. There are a number of algorithms for finding strong components in graphs. My favorite is the Tarjan Algorithm, nicely described here: Tarjan Algorithm description and pseudo code. You will need a graph class to implement this. You will find one in the Repository: C++ graph class.
- **Display:**
  Uses information in the Analyzer's AST and TypeTable to build an effective display of the dependency relationships between all files in the selected file set.
- **NoSqlDb:**
  Stores and retrieves dependency information in a NoSqlDb<std::string>, you developed in Project #1. Here, child relationships denote compile dependencies.
- **TestExecutive:**
  Provides code to demonstrate that you meet all requirements.

You will probably not need to change any of the code analyzer, other than to remove a few of its parts. You will analyze dependencies based on types and global functions you find in the analyzer's Abstract Syntax Tree (AST), then checking to see which files, if any, contain those tokens.

Requirements:

Your DependencyAnalyzer Solution:

1. **Shall** use Visual Studio 2015 and its C++ Windows Console Projects, as provided in the ECS computer labs.

2. (**1**) **Shall** use the C++ standard library's streams for all I/O and new and delete for all heap-based memory management[1].

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"[2] as illustrated in the package structure described above.

4. (**3**) **Shall** provide a TypeAnalysis package that identifies all of the types defined in a specified file. It is expected that you will build on the analysis machinery provide in the CodeAnalyzer.

5. (**5**) **Shall** provide a DependencyAnalysis package that identifies all of the dependencies between files in a specified file collection.

T N P B

6. (**4**) **Shall** find strong components in the dependency graph defined by the relationships evaluated in the previous requirement.

7. (**2**) **Shall** write the analysis results, in XML format, to a specified file.

8. (**3**) **Shall** process the command line, accepting:
   - Path to the directory tree containing files to analyze.
   - List of file patterns to match for selection of files to analyze.
   - Specification of the XML results file, supplying a default if no specification is provided.

9. (**5**) **Shall** include an automated unit test suite that demonstrates you meet all the requirements of this project[3].

---

1. 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.

2. https://en.wikipedia.org/wiki/Single_responsibility_principle

3. This is in addition to the construction tests you include as part of every package you submit.

What you need to know:

In order to successfully meet these requirements you will need to know:

1. Details of the C++ language: http://CppReference.com.

2. All those things you learned while developing code for Projects #1.

3. How the CodeAnalyzer works. The TAs and I will give you a lot of help with this.
   Also, you should look at the blog: Parser.

4. A Strong Component Algorithm: Tarjan Algorithm description and pseudo code

5. The STL Containers.

Back Back