Project #2 – Code Similarity Analysis

due date extended to Friday, March 21

version 1.2

Purpose:

This project requires you to analyze the similarity of code within a set of C++ source files. Similarity is based on the scope analysis you implemented in Project #1. Suppose that several lines or more of code have been copied from one file and pasted into another. These cut and paste operations are sometimes used when developing code that contains functionality similar to that in another file.

This duplication will help a developer get code working quickly, but causes maintenance difficulties because there are now two or more pieces of code to understand and fix if there are latent errors in the copied code. Once we detect very similar code in multiple packages we can create a function to be called by all of the other code, at the site of their duplications, allowing us to improve maintainability of all the affected packages.

Requirements:

Your Code Similarity Analysis project:

1. **shall** use standard C++[1] and the standard library, compile and link from the command line, using Visual Studio 2013, as provided in the ECS clusters and operate in the environment provided there[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. **(1) shall** identify a set of files for analysis by supplying, on the command line, a path, one or more file patterns, and a switch /S which, if present, indicates that the entire directory tree rooted at the path is searched for matching files. If the switch is not present on the command line only the directory at that path is searched.

4. **(2) shall** compare the code in all the files looking for duplicated and similar code occurring in more than one place. We do this by constructing a collection of scopes, as found with the facilities of Project #1, ordered by size, and looking for similar regions in the collections for two files being compared[3]. Note that we are not looking for identical code, but code that has the same scope structure by size and type. Thus we are finding highly similar code which may have only minor differences. That may allow us to replace the code duplicates with a common function[4].

5. **(2) shall,** for each pair of files with similar code, report the two file names and similar code regions by line number. This requirement does not include the accuracy and performance scores (see below).

6. **(1) shall** provide an option that displays the differences between the reported similar code regions.

7. **(3)** We will be deducting points for speed of execution, frequency of failures to detect, and false alarms when running on our test code. The fastest third of the submissions will lose no points for performance, the middle third will lose one point, and the slowest one third will lose 3 points. Your program **shall** time the execution of your analysis. You should insure that your display is done after all analysis and not included in your timing, as I/O is fairly slow. If you do not time the entire analysis you will lose all 3 points.

8. **(10)** You will lose 1 point for each failure to detect code that has been cut from one place and pasted into another (there will be several of these) up to a maximum of 5 points. You will lose 1 point for each false alarm, reporting similar code to code that is actually different, up to a maximum of 5 points.

9. **(1) shall** provide a test executive package and a display package, that, combined with the analysis facility, demonstrates you meet all the requirements of this specification. It is important that your demonstration is accurate, complete, and clear. Your score for meeting requirements will be based on this display. You will get no credit for requirements met but not accurately demonstrated.

---

[1] This means, for example that you may not use the .Net managed extensions to C++.

[2] VC++ 2012 is available in all the ECS clusters.

[3] This operation is similar to that used for genome sequencing.

[4] You may do the analysis for Requirement #4 anyway you wish. You must satisfy the first sentence, but the remaining sentences are suggestions, but not requirements.

Finding parts of source code files that have a high degree of similarity is very useful for code refactoring. The idea is to implement a functionality in only one place so there is only one piece of code to understand and fix if there are any latent errors in its implementation.